Lüers

MSX Gráficos y Sonido

UN LIBRO DATA BECKER

EDITADO POR FERRE MORET, S.A.

Lüers

MSX Gráficos y Sonido

UN LIBRO DATA BECKER
EDITADO POR FERRE MORET, S.A.

Este libro ha sido traducido por Dña. Petra Scheffler, arquitecta técnica y experta conocedora del sistema MSX.

Imprime: **APSSA**, ROCA UMBERT, 26 - L'HOSPITALET DE LL. (Barcelona)

Depósito legal B-35.773/85

ISBN 84-86437-25-3

Advertencia importante

Los circuitos, procedimientos y programas reproducidos en este libro, son divulgados sin tener en cuenta el estado de las patentes. Están destinados exclusivamente al uso amateur o docente, y no pueden ser utilizados para fines comerciales.

Todos los circuitos, datos técnicos y programas de este libro, han sido elaborados o recopilados con el mayor cuidado por el autor y reproducidos utilizando medidas de control eficaces. No obstante, es posible que exista algún error. FERRE MORET,S.A. se ve por tanto obligada a advertirles, que no puede asumir ninguna garantía, ni responsabilidad jurídica, ni cualquier otra responsabilidad sobre las consecuencias atribuibles a datos erróneos. El autor les agradecerá en todo momento la comunicación de posibles fallos.

INDICE
======

Introducción

I. Los MSX son ordenadores gráficos con un juego de comandos muy interesante....................1

SCREEN..1
OPEN..14
VDP...22
BASE..42

II. Por un lado existen PEEK y POKE, pero los MSX disponen además de VPEEK y VPOKE..............67

VPEEK...67
VPOKE...72

III. Con el BASIC MSX las dos pantallas de texto permiten realizar cosas interesantes.................82

CLS...82
KEY ON/OFF....................................86
WIDTH...91
COLOR...97
TAB...103
LOCATE..108
POS...116
LPOS..121
CSRLIN..124

IV. De hecho nuestro ordenador MSX es ignorante: sólo distingue 0 y 1 ... nada más.....................127

CHR$..127
ASC...133

V. Un capítulo lleno de puntos, líneas, círculos sprites y superficies ... los comandos gráficos del MSX....................................140

DRAW..140
PSET..157
PRESET......................................160
POINT.......................................164
LINE..169
CIRCLE......................................181
PAINT.......................................189
SPRITES$....................................195
PUT SPRITE..................................204

VI. Queremos completar nuestros programas con música y ruidos........................211

PLAY..211
PLAY(N).....................................224
SOUND.......................................230
BEEP..238

VII. Los ports de joystick pueden servir para diversos fines de control a través del BASIC MSX....241

STICK.......................................241
STRIG.......................................247
PAD...251
PDL...257

VIII. El programa BASIC se ejecuta tranquilamente, mientras se presenta el lenguaje máquina...........260

SPRITE ON...................................260
STRIG(N) ON.................................264
ON SPRITE GOSUB.............................269
ON STRIG GOSUB..............................273

IX. Interesantes programas de sonido y gráficos
así como rutinas en lenguaje máquina..................277

Programa 1 (cambio de sprites)..................277
Programa 2 (SCREEN 0 - 13 pantallas)............280
Programa 3 (dos juegos de caracteres)...........283
Programa 4 (sprites = letras)...................288
Programa 5 (sólo 4 sprites por línea)...........291
Programa 6 (juegos de caracteres en
líneas DATA)........................294
Programa 7 (cambio de sprites 2)................298
Programa 8 (música de piano)....................302
Programa 9 (editor PLAY con grabación).........310
Programa 10 (editor SOUND con grabación).......316
Programa 11 (editor gráfico)....................325
Programa 12 (imprimir juego de caracteres).....335
Programa 13 (gráficos en pantallas de texto)...339
Programa 14 (generador de caracteres)..........346

FRETY (un juego con magníficos gráficos).......367
MR MINER (un juego con obstáculos)..............390

Lenguaje máquina (gráficos).....................416

X. Direcciones del sistema operativo MSX
referidos a sonido y gráficos.......................421

XI. Importantes direcciones de memoria para VDP..........431

XII. Diversos juegos de caracteres en la imagen..........434

Juego de caracteres MSX.........................434
Juego de caracteres del ordenador..............456

Muy apreciados y queridos amigos interesados en el MSX,

Hasta este momento, todos los particulares que tras largas reflexiones decidieron adquirir un ordenador doméstico o Homecomputer observaban, no sin cierta envidia, a las computadoras mayores; el motivo era, sin duda, que entre tales equipos se fabricaron hace algunos años estándars formidables. Nombres como 'CP/M' y 'MS-DOS' así como la expresión 'compatibilidad IBM' garantizaron, y actualmente en mayor medida, que toda la gama de programas del ordenador X también funcionase, y sin ningún tipo de problemas, con el ordenador Y. De esta forma desaparece el problema a la pregunta '¡¿por qué no puedo utilizar este banco de datos precisamente con mi ordenador?!'.

¿Qué ocurrió y qué ocurre aún hoy con la mayoría de los ordenadores domésticos? El fabricante X lanza al mercado un ordenador que funciona sólo con los accesorios y programas de este mismo fabricante. De esta forma, la compra del ordenador X también conlleva normalmente un considerable negocio ulterior por parte de ese preciso fabricante, por lo que algunos ya han conseguido una nariz de oro. Cuando el mercado está medianamente saturado de este producto X, el mismo fabricante lanza un producto Y. Y para que vuelva a funcionar el negocio de los accesorios, se instalan simplemente nuevas conexiones en el ordenador (que tampoco corresponden a ningún tipo estándar), y además -¿cómo si no podría prosperar el pobre fabricante?- el software del producto X ya no es compatible con el producto Y. Quiere esto decir que este fabricante no se dará alguna vez de narices ... ya lo veremos. Los ordenadores MSX no sólo pueden sino que deben salir airosos al final del camino, porque aquí ya no se pretende considerar a nadie como necio.

¿Cómo puede lograrse este objetivo? Todas las empresas japonesas del ámbito de HIFI-Vídeo se reunieron hace algún tiempo y deliberaron sobre la forma de burlarse estrepitosamente de las grandes potencias económicas de los EE.UU. y Europa (ver también el desarrollo de las cámaras, el sector del vídeo y alta fidelidad así como de la

producción automovilística).
Puesto que no serían los primeros en formar parte del mercado de los Homecomputers, tenían que idear juntos algo muy especial. Finalmente se pusieron de acuerdo en la producción de ordenadores domésticos que no sólo ofrecieran compatibilidad (capacidad de intercambio) software, sino también compatibilidad hardware (conexiones a otros dispositivos).

Se puso todo ello bajo el tenor 'MSX', abreviación correspondiente al BASIC más eficaz y que procede de nuevo de 'MICROSOFT', una prestigiosa firma americana de software: 'Microsoft Super Extended Basic'.

Sin embargo, el estándar del MSX todavía abarca más: no sólo el BASIC es el mismo para todos los ordenadores MSX, sino que también sus chips de sonido y gráficos (sobre cuyas capacidades hablaremos largo y tendido en este libro) son idénticos. Además, tal y como se ha podido observar hasta ahora, cada ordenador MSX dispone de una interfase Centronic estándar; la utilización de programas de CP/M comerciales (¡de hecho este software procede de los ordenadores más grandes!) es posible en un 100%, pero eso no es todo: el sistema operativo de diskettes del MSX, el DOS-MSX, es compatible con el DOS-MS de IBM en cuanto a los datos ... ¡¡Esto significa que usted puede llevarse a casa sus datos de la oficina (¡Cuidado! ¡Protección de datos!) y continuar trabajando tranquilamente con su ordenador MSX y una estación de diskettes!!

Ya ve, pues, que esta reflexión de los japoneses tiene pies y cabeza. Esta idea ha de tener éxito forzosamente, no cabe la menor duda.

Añadimos otros dos puntos más:

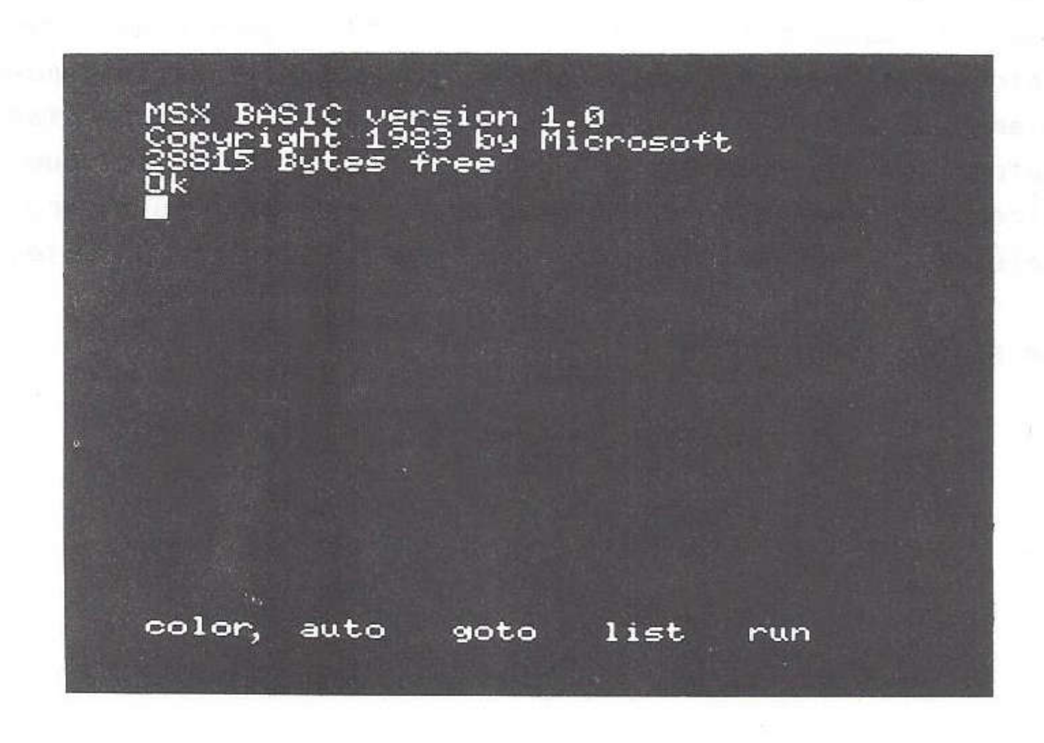
1) Puesto que casi todos los fabricantes de ordenadores MSX proceden del ramo del Vídeo y HIFI, la conexión de los Homecomputers MSX por ejemplo con un vídeo, tocadiscos de imágenes, aparatos magnetofónicos, etc. ya no está lejos de

conseguirse (ya se pudieron ver gran cantidad de ejemplos sobre ello en la HIFIvideo de Düsseldorf de 1984). Es decir: un sistema Homecomputer integrado realmente a la vida casera y que rápidamente se incorpora a su entorno.

2) ¡No sólo las posibilidades de aplicación de los ordenadores MSX son convincentes! También los datos técnicos como la ampliación a 256 KB de RAM, 32 sprites , 16 colores, gráficos de color en alta resolución de 256*192 puntos y un incorporado procesador de sonido de tres voces pueden convencer al buen conocedor de ordenadores.

Esperemos la evolución del mercado. Los japoneses (representados ahora por más de veinte empresas de renombre tales como SONY, SPECTRAVIDEO, PANASONIC, JVC, etc.), los coreanos (GOLDSTAR y SAMSUNG entre otras) y también PHILIPS de Alemania esperan que ya en 1985 gran parte de todos los Homecomputers vendidos lleven las siglas MSX. ¡Imitemos al extranjero europeo! ¡Suerte, MSX!

P.D.: Actualmente las expresiones 'ordenador doméstico' (Homecomputer) y 'ordenador personal' (Personalcomputer) son tan insignificantes que hoy en día podemos considerar a todos los ordenadores MSX como ordenadores personales.



¡Muy estimada lectora!
¡Muy estimado lector!
¡Muy estimada programadora!
¡Muy estimado programador!

Sin largos prólogos: le recibo calurosamente al gran círculo (internacional) de usuarios MSX y naturalmente como propietario orgulloso y curioso del presente libro DATA BECKER.

En las tapas y por el índice ya ha podido saber lo que le espera en este libro. Aquí y en este lugar sólo queda decir tanto como:

No ha sido fácil escribir un libro que sirviese tanto a principiantes como a profesionales, por lo que he intentado encontrar el camino intermedio. Así por ejemplo, en casi todas las descripciones de las instrucciones MSX se dan referencias sobre las correspondientes variables de sistema, interesantes para el principiante pero vitales para el profesional.

Espero que disfrute del contenido de este libro. ¡Y no sólo eso! Quizás pronto se encuentre este libro siempre al lado de su ordenador MSX y se convierta en una útil obra de consulta.

Sea cual sea el uso que le confiera a este libro, en todo caso le deseo que lo disfrute. Si tiene alguna duda o desea informarme personalmente sobre cualquier cuestión, puede llamar al 0251/47478, donde se encontrará con un contestador automático que responderá a su llamada. Por favor indique su dirección completa o en caso urgente también su número de teléfono. Intentaré informarle lo más rápidamente posible.

Le saluda atentamente

el autor del presente libro DATA BECKER.

Comandos Gráficos SCREEN

Modifica el SCREEN y diversos procesos de entrada/salida

vea también los programas del apéndice: SCREEN 1
SCREEN 2
SCREEN 3
SCREEN 4
SCREEN 5
SCREEN 6
SCREEN 7
SCREEN 8
SCREEN 9
SCREEN 10

El comando SCREEN es en cierto modo un comando central del BASIC MSX, porque gracias a él no sólo se puede cambiar entre diversas pantallas de texto y gráficas, sino que además permite aprovechar de múltiples y diversas formas los periféricos conectados al ordenador MSX.

El comando SCREEN, incluyendo su posible número de parámetros, puede utilizarse de diversas formas, como llamar tan sólo aquellas instrucciones que nos parecen útiles en ese momento, saltándose las demás (parámetros anteriores) entrando una coma.

La instrucción SCREEN se construye de la siguiente forma:

```
SCREEN A,B,C,D,E
```

Hablemos ahora detalladamente de los diversos parámetros y de sus efectos:

A: Este parámetro es con toda seguridad el más importante de la instrucción SCREEN, puesto que permite ir cambiando entre los diversos 'screens', sean gráficos o de texto.

El parámetro A puede tomar valores entre 0 y 3, donde 0 y 1 sirven en primer lugar para la representación de textos, mientras que 2 y 3 se encargan principalmente de la representación gráfica:

SCREEN 0: sólo texto, de 1 a 4 caracteres en 24 líneas
SCREEN 1: texto en 24 líneas de 1 a 32 caracteres, se pueden representar sprites
SCREEN 2: gráfico de alta resolución con 256*192 puntos, se pueden representar sprites
SCREEN 3: representación gráfica posible con 64*48 puntos, se pueden representar sprites

Así distinguimos entre pantallas de texto (SCREEN 0 y SCREEN 1) y pantallas gráficas (SCREEN 2 y SCREEN 3), a las que podemos acceder ayudados por la instrucción SCREEN y utilizando el primer parámetro.

Sin embargo, todavía falta por resolver la representación de gráficos en la pantalla de texto, así como la de texto en la pantalla de gráficos. Puesto que el procedimiento no parece muy claro, mejor será tratar el tema con cierta profundidad:

a) Gráficos en la pantalla de texto:

Con ayuda de los caracteres gráficos disponibles (por ejemplo cuadrados) y algo de fantasía, podremos dibujar pequeñas figuras. Si quiere visualizar toda la gama de caracteres gráficos a su disposición, tiene dos opciones: pulsar simultáneamente la tecla GRAPH junto con cualquier letra del teclado, o bien proyectar los caracteres gráficos en la pantalla utilizando la función CHR$. Puesto que esta última es la opción más utilizada en los programas de este libro (por su identificación absoluta de caracteres), analicemos los caracteres gráficos del MSX más detenidamente:

vea listado de SCREEN 1 del apéndice

Sin embargo, no estamos obligados a limitarnos simplemente a

este reducido juego de caracteres gráficos. También puede definir letras, números o cualquier tipo de caracteres que puedan ser representados gráficamente. Para ello sírvase de la dirección BASE de la memoria, la cual nos indica el comienzo del carácter memorizado:

PRINT BASE(2)

Resultado: 2048

Puesto que cada carácter definido se compone de 8 líneas, deberemos recurrir a un truco, por ejemplo para cambiar el aspecto del espacio (CHR$(32)), que normalmente cubre gran parte de la pantalla:

PRINT VPEEK(BASE(2)+(32*8)+1)

Con ayuda del comando VPEEK investigamos la memoria RAM de vídeo en el lugar que nos interesa en este momento. El valor 0 nos indica que el espacio se compone al menos de una línea vacía (=0). ¿Qué ocurriría si intentásemos modificar esta posición de memoria?

VPOKE BASE(21),(32*8)+1,255

Si ya ha entrado esta instrucción no se sorprenderá del siguiente resultado: inmediatamente después de ejecutarla se modifica gran parte de la pantalla. ¿Qué hemos hecho? Encargamos a nuestro "colega el ordenador MSX" que coloque 255 = una barra blanca, en la posición de memoria correspondiente al espacio.

Sin embargo, puesto que en la pantalla de texto sólo pueden representarse 256 caracteres diferentes, cada espacio (uno de los 256 caracteres) ha cambiado instantáneamente según la modificación del código de la entrada.

Puede eliminar este efecto más bien molesto en trabajos posteriores colocando otra vez con VPOKE el valor inicial en la posición de memoria modificada. Pero también puede cargar nuevamente los caracteres originales en cada momento entrando cualquier pantalla SCREEN y el primer parámetro.

Sin embargo, será casi imposible -a menos que de entrada prescindamos completamente de la representación simultánea de texto y gráficos en SCREEN 0 o SCREEN 1- dibujar imágenes en una pequeña parte de la pantalla de texto (en total una superficie de hasta 256*192 puntos) según la forma antes descrita. Ello se muestra de modo casual en el programa siguiente:

vea listado de SCREEN 2 del apéndice

Después de esta impresionante modificación de la pantalla, entre otra vez SCREEN 0 para que todo vuelva a su estado normal (desgraciadamente en este caso no podrá reconocer la entrada SCREEN en la pantalla; haga la entrada a ciegas y a continuación pulse la combinación de teclas CTRL-E (para borrar hasta el final de la línea)).

Esperamos que esta limitada representación gráfica en la pantalla de texto SCREEN 0 no le presente más dudas; pero queremos advertirle en este lugar, que debería evitar la llamada de comandos gráficos directos (tales como CIRCLE, PAINT, LINE), mientras se encuentre en la pantalla de texto. En caso contrario, se emitirá el mensaje de error 'Illegal function call'.

b) Texto en pantallas gráficas

¿Qué ocurre ahora con el texto en las pantallas gráficas? Una sencilla instrucción PRINT no será suficiente en los ordenadores MSX:

vea listado de SCREEN 3 del apéndice

Más bien tendrá que abrir antes un canal de datos (con OPEN) para poder escribir en las pantallas gráficas y enviar después la instrucción PRINT≠ directamente hacia este canal, tal como se muestra en el siguiente programa:

vea listado de SCREEN 4 del apéndice

En la línea 30 de este programa se indica al ordenador MSX el canal de datos a través del cual deberá efectuarse la salida hacia la pantalla gráfica. Después de ello tendrá la oportunidad -ya que gráficamente se crea punto por punto- de ocultar la primera imagen escrita con un segundo texto o incluso un gráfico. Sólo pueden aparecer problemas si desea trabajar simultáneamente con distintos colores.

Utilizando el comando DRAW -de forma parecida a la instrucción LOCATE para posicionar el cursor en la pantalla de texto- es posible posicionar el texto pixel por pixel en el lugar de la pantalla gráfica que nos parece especialmente razonable; dibujemos por ejemplo con el siguiente programa la combinación de letras 'MSX' en forma de curva sinusoidal repetidas veces en la pantalla:

vea listado de SCREEN 5 del apéndice

Al igual que SCREEN 2, SCREEN 3 ofrece la posibilidad de representar gráficos y textos de colores. Sin embargo, hay que tener especial precaución al evaluar correctamente el tamaño de los caracteres representados. Por otra parte, las coordenadas se aplican de igual forma que en SCREEN 2.

Otra advertencia:

Los lectores muy atentos que ya conocen el comando DRAW se formularán ahora dos preguntas que queremos responder brevemente:

1) ¿Podemos representar en la pantalla letras y signos de medidas grandes o pequeñas con ayuda de la instrucción DRAW "S"?

Respuesta: no. Para ello se requiere un juego de caracteres definido nuevamente con ayuda de la instruccion DRAW, puesto que el comando DRAW del programa anterior sólo se refiere al posicionamiento exacto del cursor gráfico, pero no a la forma de los caracteres emitidos.

Sin embargo, existe una posibilidad de representar el texto de forma ampliada y/o invertida en las pantallas gráficas. Para este fin leemos con POINT los puntos en la pantalla y con PSET volvemos a emitir inmediatamente la representación modificada.

vea listado de SCREEN 6 del apéndice

2) Puesto que al activar SCREEN 0 se cambia al juego de caracteres estándar, su configuración debería encontrarse todavía en al memoria (ROM).

Respuesta: correcto. Respecto a ello observe la correspondiente sección de ROM en representación binaria utilizando el siguiente programa:

vea listado de SCREEN 7 del apéndice

La memoria de video RAM contiene además una copia del mismo juego de caracteres (copiado de las anteriores posiciones de memoria ROM) al utilizar SCREEN 0:

vea listado de SCREEN 8 del apéndice

La representación de texto en la pantalla gráfica SCREEN 2 se parece sólo exteriormente a la reproducción en las pantallas de texto: por un lado no podemos predeterminar la cantidad de caracteres por línea de pantalla en la página gráfica (en pantallas de texto posible con WIDTH), sino que siempre disponemos de 32 caracteres por línea en caso de no activar letra por letra utilizando DRAW "B M"; por otro lado no se efectúa el scroll de la pantalla hasta que la línea 24 no se ha llenado. Cuando se cumpla este último requisito, su

ordenador MSX comienza nuevamente a escribir en la primera línea encima del texto que se encuentra allí, sin borrarlo antes. Además es muy complicado volver a borrar el texto de la pantalla gráfica sin borrar también la totalidad de la pantalla: podemos dibujar un cuadrado del tamaño del texto a borrar y rellenarlo del color de fondo (función LINE ...,,BF), o bien podemos escribir otra vez el mismo texto exactamente en la misma posición de la pantalla, pero utilizando el color de fondo; finalmente, también podemos intentar poner de nuevo la sección de texto deseada de color del fondo con VPOKEs dirección por dirección en la RAM de vídeo (este es sin duda el camino más complicado).

Para finalizar la explicación del parámetro 1 del comando SCREEN, observe cómo se puede borrar, volver a escribir, borrar otra vez, ... con ayuda de un cuadrado relleno. Al borrar y volver a escribir se produce finalmente un efecto intermitente:

vea listado de SCREN 9 del apéndice

Volvamos ahora a los demás parámetros de la instrucción SCREEN:

B: Si queremos dirigirnos únicamente a este parámetro a través del comando SCREEN, debemos indicar una coma en lugar del primer parámetro saltado, por ejemplo:

```
SCREEN ,1
```

Con el parámetro B indicamos la configuración de sprites en la pantalla. Para ello disponemos de cuatro posibilidades:

SCREEN ,0: sprites de 8*8, tamaño original de pixel
SCREEN ,1: sprites de 8*8, represent. de pixel ampliada
SCREEN ,2: sprites de 16*16, tamaño original de pixel
SCREEN ,3: sprites de 16*16, repres. de pixel ampliada

Es necesario que usted decida el tamaño que deberán tener los sprites en la pantalla (formato de 8*8 ó 16*16, representado en tamaño normal o ampliado) cuando esté planificando el programa, puesto que sólo dispone siempre de un único tamaño para todos los sprites definidos.

Para que tenga la posibilidad de investigar esto a fondo, defina simplemente de entrada un sprite de 16*16 con ayuda de la instrucción SPRITE$. Todo aquello que no sirva para su ordenador MSX en el tamaño de sprites escogido -por ejemplo al escoger SCREEN 0 se muestra sólo la esquina superior izquierda del propio sprite de 16*16- sencillamente no será emitido en pantalla.

Al cambiar la programación de un tamaño de sprite a otro distinto, se borran todas las representaciones actuales de sprites de la pantalla y también todas sus definiciones de la memoria (VRAM), cuya distribución habrá que volver a reorganizar (naturalmente el ordenador MSX efectúa esta redistribución automáticamente).

El siguiente programa le muestra los diversos tamaños de sprites juntos:

vea listado de SCREEN 10 del apéndice

Cabe mencionar además (como también se indica junto al comando SPRITE$) que puede representar simultáneamente hasta 32 sprites diferentes en la pantalla, sea cual sea su tamaño definido. Sin embargo, si utiliza los sprites más pequeños (de 8*8 o bien de 8*8 ampliado) podrá almacenar simultáneamente hasta 256 diferentes configuraciones de sprites en la memoria. Si utiliza, en cambio, los sprites más grandes (que en realidad se componen de 2*2=4 pequeños sprites 8*8) de 16*16 puntos aislados, podrá almacenar 'sólo' 64 configuraciones diferentes de sprites en la bien definida tabla de generadores de sprites del VRAM.

C: Para permitirle tener un control auditivo además del táctil y visual al pulsar cualquier tecla de su ordenador MSX, al igual que en los mayores ordenadores personales, se emite un suave chasquido con cada pulsación efectuada en el teclado (al producirse algún error, en cambio, suena un fuerte BEEP).

Si le molestase este continuo chasquido, podrá evitarlo entrando simplemente

```
SCREEN ,,0
```

(en este caso nos saltamos los dos primeros parámetros de SCREEN). Naturalmente podrá volver a conectarse en cualquier momento poniendo un valor entre 1 y 255 en este tercer parámetro, tal como le muestra el siguiente ejemplo:

```
SCREEN ,,1 (conectar chasquido del teclado)
SCREEN ,,0 (desconectar chasquido del teclado)
```

D: Si después de realizar sus primeros pasos de programación desea grabar los resultados conseguidos (al menos una línea de programa), podrá efectuarlo tanto con un cassette (cualquiera, pero que debe poder conectarse al ordenador MSX mediante los correspondientes cables de conexión) como con una unidad de disco (un pequeño tocadiscos que no utiliza ningún zafiro, sino que lee y/o graba los datos magnéticamente con ayuda de un dispositivo parecido a un cabezal de sonido).

La carga o grabación efectuada con cassette requiere cierto tiempo en su ordenador MSX (con un programa de juego algo extenso, tal como el añadido en el apéndice, puede tratarse incluso de varios minutos). Puede escoger entre dos opciones diferentes: o tiene un cassette muy bueno y utiliza buenas cintas de ferro para grabar o cargar programas, en cuyo caso debe indicarle con

```
SCREEN ,,,2
```

a su ordenador MSX que tendrá que grabar con velocidad doble (o sea 2400 baudios = 2400 bits por segundo = aproximadamente 300 caracteres por segundo); en caso contrario (aunque tampoco se recomienda utilizar precisamente los peores cassettes o cintas para cargar/grabar), escriba la instrucción

SCREEN ,,,1

antes de grabar/cargar. Ahora los datos se almacenarán a velocidad normal en el cassette (o sea 1200 baudios = 1200 bits por segundo = aproximadamente 150 caracteres por segundo).

¿Cómo ocurre esto? Los datos fluyen, dicho sencillamente, a doble velocidad hacia el cassette. ¡No piense acaso que su cassette normal y corriente tendrá que sintonizarse ahora a velocidad turbo!

Vayamos finalmente con el último parámetro que nos ofrece la instrucción SCREEN:

E: Desde hace años existe un acuerdo internacional entre los diversos fabricantes de ordenadores sobre la conveniencia de conseguir una compatibilidad entre todos los sistemas. Con este fin, las empresas ASCII (por el juego de caracteres) y MICROSOFT (por el BASIC) se han ganado hasta ahora un nombre en el tan reñido mercado de los solitarios.

Pero su ordenador MSX dispone de otro grado más: su BASIC es compatible (cien por cien) con los demás ordenadores MSX; pero eso no es todo: las conexiones de los periféricos son idénticas ... prácticamente todo es idéntico, con algunas pocas excepciones.

¿Qué tiene que ver todo eso con la explicación del parámetro E? Los caracteres 32 a 127 de su ordenador MSX se corresponden completamente con la norma ASCII. Estos caracteres (cada letra, número y carácter especial) pueden

imprimirse tanto en una impresora especial de MSX como en cualquier otra impresora estándar sin ningún tipo de problemas (suponiendo que se dispone de la adecuada conexión por cable).

Además (a partir de 128 caracteres) su ordenador MSX dispone también de diversos caracteres especiales y caracteres gráficos adicionales. Si se conecta una impresora MSX a su ordenador MSX tampoco tendrá problemas al impimir estos caracteres, suponiendo que se haya colocado un 0 en el parámetro E de la instrucción SCREEN:

```
SCREEN ,,,,0
```

Si conecta una impresora que no sea MSX, los caracteres especiales a partir de CHR$(128) deberán emitirse en forma de espacios para evitar confusiones en la impresora. En este caso deberá colocar un valor mayor de 0 (de 1 a 255) en el lugar del parámetro E de la instrucción SCREEN, y los caracteres especiales serán sustituidos por espacios, que puede rellenar con grafismos realizados a mano si lo desea.

SCREEN 1

```
10 COLOR 1,15
20 SCREEN 1
30 FOR N=192 TO 255
40 PRINT CHR$(N);
50 NEXT N
```

SCREEN 2

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=0 TO 959
40 PRINT CHR$(INT(RND(1)*223)+32);
50 NEXT N
60 FOR N=2048 TO 4095
70 VPOKE N,INT(RND(1)*255)
80 NEXT N
```

SCREEN 3

```
10 COLOR 1,15
20 SCREEN 2
30 PRINT "Ordenador MSX"
40 GOTO 40
```

SCREEN 4

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 PRINT #1,"Ordenador MSX"
50 CLOSE
60 GOTO 60
```

SCREEN 5

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 FOR N=1 TO 255 STEP 3
50 Y=SIN(N/20)*50+100
60 DRAW "bm =n;,=y;"
70 PRINT #1,"MSX";
80 NEXT N
90 CLOSE
100 GOTO 100
```

SCREEN 6

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "bm 0,0"
50 PRINT #1,"MSX"
60 FOR N=0 TO 23
70 FOR M=0 TO 7
80 A=POINT(N,M)
90 PSET(N+50,M+50+2),A
```

```
100 Z-Z+1
110 PSET(N+50,M+50+Z),A
120 NEXT M
130 Z=0:NEXT N
140 GOTO 140
```

SCREEN 7

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 FOR N=&H1BBF TO &H1BBF+2047
50 PRINT STRING$(8-LEN(BIN$(PEEK(N))),"0
");BIN$(PEEK(N))
60 NEXT N
```

SCREEN 8

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 FOR N=2048 TO 2048+2047
50 PRINT STRING$(8-LEN(BIN$(VPEEK(N))),"
0");BIN$(VPEEK(N))
60 NEXT N
```

SCREEN 9

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "bm 128,96"
50 PRINT #1,"MSX"
60 FOR N=1 TO 200
70 NEXT N
80 LINE(128,96)-(152,104),15,BF
90 FOR N=1 TO 200
100 NEXT N
110 GOTO 40
```

SCREEN 10

```
10 COLOR 1,15
20 OPEN "GRP:" FOR OUTPUT AS #1
30 FOR N=0 TO 3
40 SCREEN 2,N
50 DRAW "bm 50,50"
60 SPRITE$(1)=STRING$(32,"U")
70 PRINT #1,"Tamano";N
80 PUT SPRITE 1,(128,96)
90 FOR M=1 TO 1000
100 NEXT M
110 NEXT N
120 GOTO 30
```

Comandos Gráficos OPEN

Abre canales de datos por ejemplo para emitir texto en páginas gráficas

vea también los programas del apéndice: OPEN 1
OPEN 2
OPEN 3
OPEN 4
OPEN 5
OPEN 6
OPEN 7
OPEN 8
OPEN 9
OPEN 10

Con ayuda del comando OPEN podemos abrir canales de datos en general, tanto si la entrada/salida indicada a continuación se efectúa de/hacia cassette, impresora, pantalla gráfica o de texto como en unidad de disco. Las abreviaciones que corresponden a cada medio de entrada/salida (que se realiza carácter por carácter, o sea secuencialmente) son las siguientes:

CAS: para cassette
LPT: para impresora (Line-PrinTer)
CRT: pantalla de texto
GRP: para texto en pantalla gráfica
1: ó 2: para unidades de disco

Puesto que nos ocupamos aquí de los temas gráficos y sonido trataremos tan sólo con los medios de salida adecuados en este sentido en cuanto se refiere a la impresora, la pantalla de texto y la pantalla gráfica:

1) Salida con OPEN "LPT:" hacia la impresora o con OPEN "CRT:" hacia la pantalla de texto:

Esta instrucción OPEN no se utilizará prácticamente nunca en modo directo, porque a este fin disponemos de otras instrucciones mucho más lujosas en BASIC MSX:

PRINT "..." para salida de texto hacia la pantalla
LPRINT "..." para salida de texto hacia la impresora conectada
LIST para emitir listados de programas en la pantalla
LLIST para emitir listados de programas en la impresora conectada

Sin embargo, por su claridad gráfica le mostraremos la forma de dar la instrucción OPEN para emitir datos hacia la impresora conectada:

```
OPEN "LPT:" FOR OUTPUT AS ≠1
PRINT ≠1, "Los ordenadores MSX son extraordinarios"
CLOSE ≠1
```

y la misma salida efectuada con ayuda del comando LPRINT del BASIC MSX:

```
LPRINT "Los ordenadores MSX son extraordinarios"
```

Usted mismo verá qué opción se realiza con mayor facilidad: ¡¿utilizando los comandos OPEN, PRINT ≠ y CLOSE o bien el comando LPRINT?!

Se preguntará, ¿por qué existe entonces la posibilidad de salida a efectuar con OPEN "LPT:", PRINT ≠ y CLOSE? Muy sencillo: de esta forma podemos dirigir la salida una vez hacia el canal 1 (la pantalla de texto en el ejemplo siguiente), otra vez hacia el canal 2 (la impresora en el ejemplo siguiente):

vea listado de OPEN 1 del apéndice

De la misma manera también sería posible enviar la salida hacia canales adicionales por abrir (3 y 4), que razonablemente podrían ser CAS: (cassette) ó 1: (unidad de disco) así como hacia la pantalla gráfica (GRP:). Puesto que podemos dirigirnos a los periféricos utilizando el simple número (1, 2, 3 ó 4), es fácil emitir ahora en un programa toda la salida hacia la impresora en lugar de hacerlo hacia la pantalla. Ello representa un ahorro de tiempo y de espacio. Usted se dará cuenta de su significado cuando haya sustituido alguna vez todos los comandos PRINT por LPRINT en un programa largo (mientras que la instrucción OPEN sólo requiere cambiar el valor de una variable).

Otra pequeña diferencia de la salida hacia pantalla de texto (CRT:) o impresora (LPT:) en comparación con los comandos BASIC PRINT y LPRINT: si desea efectuar una salida formateada (escritura en colummnas para crear tablas) puede añadir una coma al comando PRINT o LPRINT. De esta forma se define un tabulador cada 14 columnas donde continuará la escritura. Si la cantidad de caracteres a emitir supera la de los caracteres que pueden representarse en la línea sin rebasarla, se continúa escribiendo en la próxima zona tabulada de la línea siguiente. Sin embargo, al utilizar OPEN "LPT:" o OPEN "CRT:" la próxima salida después de una coma se efectúa siempre 14 caracteres más adelante, independientemente de si la cantidad de caracteres a emitir sobrepasa el límite de la pantalla o no (esto varía un poco al realizar OPEN "LPT:"):

Salida hacia la pantalla utilizando PRINT:

vea listado de OPEN 2 del apéndice

Salida hacia la pantalla utilizando OPEN "CRT:":

vea listado de OPEN 3 del apéndice

Salida hacia la impresora utilizando LPRINT:

vea listado de OPEN 4 del apéndice

Salida hacia la impresora utilizando OPEN "LPT:":

vea listado de OPEN 5 del apéndice

¡Recuerde: con ayuda del comando OPEN (abrir canal para dirigirnos hacia pantalla e impresora) no se puede formatear la salida!

2) Mientras que la salida hacia impresora o pantalla de texto con OPEN "LPT:" ó OPEN "CRT" no comporta grandes diferencias comparado con los comandos LPRINT y LLIST, la salida hacia una de las pantallas gráficas es bien diferente. A tal fin intentamos emitir texto en la pantalla gráfica utilizando el comando PRINT sin abrir antes un canal de datos con OPEN:

vea listado de OPEN 6 del apéndice

Por muy bonito y claro que nuestro programa pueda parecer, en la pantalla gráfica no sucede nada. Probemos pues con OPEN "GRP:":

vea listado de OPEN 7 del apéndice

Ahora la salida no presenta ningún problema y se puede ver la letra claramente en la esquina superior izquierda.

Tal y como ya mencionamos anteriormente, debemos ir con especial cuidado al trabajar con la pantalla gráfica, puesto que aquí todo discurre bajo otros principios que en la pantalla de texto: en el caso de nuestra emisión de texto es importante saber que, de hecho, cada letra debe verse y entenderse como un dibujo de 8*8 puntos. Por esta razón podemos entrar por un lado línea por línea con distancias normales de caracteres y líneas (igual que en SCREEN 1) mientras que disponemos de 32*24 caracteres:

vea listado de OPEN 8 del apéndice

o situamos nuestra letra en el lugar exacto (escogiendo la posición de la pantalla punto por punto) deseado (no dependiendo del campo de matriz 32*24):

vea listado de OPEN 9 del apéndice

Puesto que en las pantallas gráficas no disponemos del comando LOCATE, debemos servirnos del comando DRAW "B M" (dirígete ciegamente con el cursor hacia la correspondiente posición de pantalla). Para cambiar el color del texto emitido debemos entrar un COLOR adecuado a pesar del lujoso comando DRAW.

¡Pero por favor no se haga demasiadas ilusiones! Los caracteres emitidos no pueden girarse (DRAW "A") ni ampliarse (DRAW "S") de esta forma tan sencilla.

Otra advertencia: como ya hemos dicho antes y demostrado también en programas, la pantalla gráfica trabaja de forma muy distinta a las pantallas de texto SCREEN 0 o SCREEN 1. Así que, al emitir texto en la pantalla gráfica, deberá actuar con precaución cuando llegue a la última línea de pantalla: aquí no se efectúa el scroll hacia arriba, sino que se escribe de nuevo encima del texto comenzando por la esquina superior izquierda. Hay dos posibilidades de limpiar la pantalla:

1) Borrado completo con ayuda de CLS

2) Borrado parcial al escribir de nuevo el mismo texto empleando el color del fondo (parecido al hecho de borrar un error en el papel con "Tipp Ex" blanco).

Para finalizar este capítulo queremos mostrarle el efecto provocado al reescribir encima del texto con el color del fondo:

vea listado de OPEN 10 del apéndice

¡La escritura encima del texto debe ser completamente

idéntica a la anterior imagen existente, para evitar que queden partes sin cubrir! Para demostrarlo cambie la línea 90 de la forma siguiente:

```
90 PRINT ≠1, "Ordenitor MSX"
```

Al ejecutar el programa modificado permanecen en la pantalla algunos restos de la palabra "ordenador" (línea 60), puesto que la palabra "ordenitor" (línea 90) no la cubre al 100%.

OPEN 1

```
10 COLOR 1,15
20 SCREEN 0
30 MAXFILES=2
40 OPEN "CRT:" FOR OUTPUT AS #1
50 OPEN "LPT:" FOR OUTPUT AS #2
60 FOR N=1 TO 2
70 PRINT #N,"Los ordenadores MSX son SOB
ERBIOS"
80 NEXT N
90 CLOSE
```

OPEN 2

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 20
40 PRINT "Magnifico MSX",
50 NEXT N
```

OPEN 3

```
10 COLOR 1,15
20 SCREEN 0
30 OPEN "CRT:" FOR OUTPUT AS #1
40 FOR N=1 TO 20
50 PRINT #1,"Magnifico MSX",
60 NEXT N
70 CLOSE
```

OPEN 4

```
10 FOR N=1 TO 20
20 LPRINT "Magnifico MSX",
30 NEXT N
```

OPEN 5

```
10 OPEN "LPT:" FOR OUTPUT AS #1
20 FOR N=1 TO 20
30 PRINT #1,"MSX es magnifico",
40 NEXT N
50 CLOSE
```

OPEN 6

```
10 COLOR 1,15
20 SCREEN 2
30 PRINT "Graficos con MSX"
40 GOTO 40
```

OPEN 7

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 PRINT #1,"Graficos MSX"
50 GOTO 50
```

OPEN 8

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 FOR N=1 TO 100
50 PRINT #1,"MSX ";
60 NEXT N
70 CLOSE
80 GOTO 80
```

OPEN 9

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 FOR N=1 TO 255 STEP 3
50 M=SIN(N/20)*50+100
60 DRAW "B M =n;,=m;"
70 PRINT #1,"MSX"
80 NEXT N
90 CLOSE
100 GOTO 100
```

OPEN 10

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "B M 128,96"
50 COLOR 1
60 PRINT #1,"Ordenador MSX"
70 DRAW "B M 128,96"
80 COLOR 15
90 PRINT #1,"Ordenador MSX"
100 GOTO 40
```

Comandos Gráficos VDP

Lee y escribe el contenido de los 9 registros del procesador de visualización

vea también los programas del apéndice: programa 2
programa 3
programa 5
programa 7
programa 14

Al utilizar este comando debemos efectuar la asignación de un valor en forma binaria, por lo que aquí queremos volver a señalar el correspondiente proceso:

Un byte se compone de ocho bits que sólo pueden tomar valor 1 ó 0. Con ayuda de la potencia de base dos se puede calcular, por ejemplo, el número decimal 255 utilizando ocho unos. Para asignar un valor binario a una variable debe indicar la combinación '&B' delante del número en cuestión.

De esta forma, la asignación del binario 00100100 a la variable A será como sigue:

A = &B00100100

El comando VDP representa un tipo de asignación de variables. El descifrado de la combinación de letras VDP (Video Display Processor) indica el tipo de variable de que se trata.

Utilizando el comando VDP seguido de una cifra de 0 a 8 situada entre paréntesis, podemos dirigirnos directamente al VDP para asignarle valores o leerlos en los registros. Deberá actuar con precaución al efectuar la entrada de valores, al menos hasta que conozca exactamente cada uno de los registros del VDP, así como la configuración general de la RAM de vídeo, puesto que el manejo del comando VDP (confío que sea suficientemente comprensible) nos introduce en las profundidades y secretos del ordenador MSX, que además de entusiasmo también puede albergar alguna sorpresa

desagradable. Sin embargo, en el peor de los casos, la programación errónea sólo puede provocar la pérdida de su programa o bien su desaparición de la pantalla. ¡Su ordenador no se estropeará nunca por una programación errónea!

Vayamos a analizar ahora los registros 0 a 8 del VDP más detenidamente:

El contenido de los registros 0 a 7 del VDP puede ser leído (PRINT VDP(N)) y escrito (VDP(N)=M), mientras que el registro 8, llamado registro de estado, es de lectura pero no de escritura.

Los contenidos actuales de los registros 0 a 7 se almacenan además en las posiciones de memoria &HF3DF a &HF3E6 para permitir su lectura de otra forma. El contenido del registro 8 también se memoriza en otra parte de la RAM, exactamente en la posición &HF3E7.

Antes de hablar detenidamente de cada registro, anteponemos en cada caso su correspondiente contenido en forma binaria, tal como se nos muestra al conectar el ordenador MSX (SCREEN 0, blanco en primer término y azul oscuro en el fondo).

Registro 0

Contenido actual llamado con:
PRINT PEEK(&HF3DF)
en forma binaria (8 bits completos): 00000000
asignación con: VDP(0)=...

En este registro únicamente los bits 0 y 1 tienen un contenido registrable, mientras que los bits 2 a 7 pueden tomar cualquier contenido sin función.

El valor de bit 0 debe ser 0, hasta que estemos en condiciones de activar un segundo VDP paralelo al VDP que en ese momento se encuentra en su ordenador MSX. Al llegar a

este punto se podría colocar un 1 en el bit 0, conectando así el segundo VDP. Ya ve: ya en la confección del concepto actualmente válido se pensó en el desarrollo del ahora estándar del MSX (para el que se garantizará naturalmente una compatibilidad).

El bit 1 trabaja junto con el registro 1 (bits 3 y 4) y hace referencia al modo SCREEN escogido en ese momento. De ello trataremos con más detalle al hablar de los bits 3 y 4 del registro 1; de momento nos limitaremos a lo siguiente: el bit 1 del registro 0 suele ser = 0. Sólo para dirigirnos al gráfico de alta resolución deberemos colocar un 1 en este lugar.

Ejemplos para demostrar el uso del registro 0:

VDP(0)=&B00000001

En caso de que exista se conecta el segundo VDP

VDP(0)=&B00000010

En el caso de que los bits 3 y 4 del registro 1 tomen valor 0, su ordenador MSX cambia automáticamente al gráfico de alta resolución = SCREEN 2.

Registro 1

Contenido actual llamado con:
PRINT PEEK(&HF3E0)
en forma binaria (8 bits completos): 11110000
asignación con: VDP(1)=...

El bit 0 tiene valor 0 si hemos decidido dirigirnos a sprites de tamaño normal en la pantalla, o sea sprites de 8*8 tienen realmente un tamaño de 8*8 pixel, mientras que los sprites de 16*16 también se componen de 16*16 puntos gráficos en la pantalla.

El bit 0 tiene valor 1 si deseamos trabajar con sprites ampliados, teniendo entonces los sprites de 8*8 un tamaño de 16*16 puntos en la pantalla, ya que cada punto se representa con un tamaño de 2*2 puntos. Lo mismo rige para los sprites de 16*16 que, en el momento en que el bit 0 del registro 1 tome valor 1, se representan ampliados en una matriz de 32*32 puntos en la pantalla.

Por cierto: en el BASIC del MSX podemos expresar el cambio del bit 0 del registro 1 activando adecuadamente el segundo parámetro de SCREEN:

SCREEN ,0: sprites de 8*8, tamaño normal
SCREEN ,1: sprites de 8*8, ampliados
SCREEN ,2: sprites de 16*16, tamaño normal
SCREEN ,3: sprites de 16*16, ampliados

Seguramente se preguntará ahora: ¿por qué activar el registro 1 del VDP y allí activar o desactivar el primer bit si podemos conseguir lo mismo más fácilmente con el comando SCREEN y su segundo parámetro? Como ya mencioné al principio de este capítulo: al trabajar con los registros del VDP nos introducimos muy profundamente en el interior del ordenador MSX. Para usted como programador de BASIC, este conocimiento no siempre es útil y razonable. Pero piense por favor en los programadores de lenguaje máquina que no pueden utilizar el comando SCREEN, sino que más bien deben trabajar con los registros de VDP y los comandos INP y OUT.

El bit 1 tiene valor 0 si deseamos trabajar con sprites de 8*8. Comparado con los comandos SCREEN, en este caso se trata de la elección de SCREEN ,0; sin embargo, también podríamos trabajar con SCREEN ,1 suponiendo que el bit 0 del registro 1 estuviese activado para ampliación = valor 1.

El bit 1 tiene valor 1 si deseamos trabajar con sprites de 16*16. Comparado con los comandos SCREEN, en este caso se trata la elección de SCREEN ,2, aunque también sería posible trabajar con SCREEN ,3 suponiendo que el bit 0 del registro 1 estuviese activado para ampliación = valor 1.

Sin embargo, el direccionamiento directo de los bits 0 y 1 del registro 1 del VDP se diferencia del comando SCREEN ,N en un punto esencial: mientras que SCREEN ,N borra las muestras de sprites definidas anteriormente del VRAM, no ocurre lo mismo al utilizar el comando VDP.

Para demostrar este fin hemos incluído en este libro un programa que le mostrará claramente el paso directo de los cuatro diferentes tamaños o resoluciones de sprites:

vea programa 7 del apéndice

El bit 2 del registro 1 no cumple ninguna función de momento. Tanto si aquí pone un 1 como un 0, el aspecto de la pantalla no cambiará en absoluto.

Los bits 3 y 4 trabajan estrechamente relacionados y no sólo eso: en sus actividades incluyen además el mensaje sobre el estado del bit 1 del registro 0.

Con estos tres bits se calcula el SCREEN deseado o bien el que de momento esté en acción. Para entender mejor los tres bits debemos referirnos a la siguiente tabla:

registro 0 bit 1	registro 1 bit 3	registro 1 bit 4	SCREEN
0	0	1	0
0	0	0	1
1	0	0	2
0	1	0	3

Con ayuda de estos tres bits podemos cambiar el SCREEN según conveniencia, por lo que este procedimiento también les debería parecer lógico sobre todo a los programadores de lenguaje máquina, puesto que con la programación de estos tres bits puede olvidarse del primer parámetro del comando SCREEN, para trabajar ya sólo con estos tres registros. En combinación con los bits 0 y 1 del registro 1, incluso podemos prescindir del segundo parámetro de SCREEN por dirigirse directamente al VDP. Pero también trabajando en

BASIC sabrá apreciar el direccionamiento directo al registro y a los bits del VDP, máximo en el momento en que decida modificar la configuración de la VRAM según propios deseos. Encontrará más información al respecto en la descripción de los registros 2 a 6.

El bit 5 desconecta la lectura automática de interrupción en lenguaje máquina si está activado a 0, y la conecta si está activado a 1.

¿Recuerda todavía los comandos ON SPRITE GOSUB y SPRITE ON/OFF/STOP? Justamente a ellos hace referencia el bit 5 del registro 1 de VDP. Si se efectuase el control de colisión de sprites en su ordenador MSX, significa que hasta ahora ha utilizado el comando SPRITE ON, con el que se efectúa un salto hacia un subprograma indicado por el comando ON SPRITE GOSUB, donde puede visualizarse, por ejemplo, una explosión en la correspondiente posición de la pantalla. Con SPRITE OFF volvió a desconectar entonces el control de colisión de los sprites. Puede obtener el mismo resultado utilizando el bit 5 del registro 1 de VDP: si este bit=1, su función corresponde al comando en BASIC SPRITE ON. Si el bit=0, su función corresponde al comando en BASIC SPRITE OFF. Sin embargo, el comando SPRITE STOP en MSX no podrá ajustarse con ayuda de un registro de VDP.

El bit 6 tiene valor 1 si deseamos trabajar habitualmente en la pantalla con colores de primer plano, de fondo y eventualmente de bordes.

Sin embargo, al poner el bit 6 al valor 0, se activa el color del primer plano de la pantalla igual que el del área de los bordes. Ello tiene el siguiente significado: ahora podrá crear en la pantalla una imagen gráfica o de texto y no conectarlo hasta su terminación, poniendo otra vez un 1 en el bit 6 del registro 1.

El bit 7 siempre debe tener valor 1, puesto que su ordenador MSX dispone de una VRAM de 16 kilobytes. Si el bit 7 del registro 1 está activado a 1, el VDP sabrá que dispone al

menos de 8 kilobytes de VRAM, y en caso normal incluso de 16 kilobytes. En anteriores versiones de MSX se tomó en consideración por lo visto que el ordenador pudiese disponer también de sólo 4 kilobytes de VRAM. En este sentido se señalaría el bit 7 del registro 1 con un 0.

Ejemplos de aplicación del registro 1:

VDP(1)=&B11100011

Bit 0 = 1: sprites ampliados (cada punto del sprite = 2*2 puntos de pantalla).

Bit 1 = 1: se representan los sprites grandes (= sprites de 16*16).

Bits 0 y 1 corresponden pues al comando en BASIC SCREEN ,3

Bit 2 = 0: sin significado.

Bit 3 = 0 y bit 4 = 0: si además el bit 1 del registro 0 = 1, hemos creado SCREEN 2 (gráfico de alta resolución) con estos 2 bits; si el bit 1 del registro 0 = 0, hemos activado SCREEN 1.

Bit 5 = 1: activado el control de interrupción para la colisión de sprites; corresponde al comando en BASIC SPRITE ON.

Bit 6 = 1: representación de pantalla normal = el gráfico se muestra en el color del primer plano.

Bit 7 = 1: disponemos de un ordenador MSX con una VRAM de 8 o 16 kilobytes.

VDP(1)=&B10001000

Bit 0 = 0: los sprites se representan en tamaño original.

Bit 1 = 0: se representan los sprites pequeños (= 8*8).

Bits 0 y 1 corresponden pues al comando en BASIC SCREEN ,0

Bit 2 = 0: sin significado.

Bit 3 = 1 y bit 4 = 0: el bit 1 del registro 0 debe tener valor 0, con lo que nos encontramos en SCREEN 3 (gráficos de bloque).

Bit 5 = 0: desconectado el control de colisión de los sprites (corresponde al comando en BASIC SPRITE OFF)

Bit 6 = 0: el gráfico se representa invisible con color del borde traslúcido; cuando volvamos a activar a 1 el bit 6 del registro 1, podremos ver también el texto o el gráfico representado.

Bit 7 = 1: disponemos de un ordenador MSX con una VRAM de 8 ó 16 kilobytes.

Registro 2

Contenido actual llamado con:
PRINT PEEK(&HF3E1)
en forma binaria (8 bits completos): 00000000
asignación con: VDP(2)=...

Sólo los bits 0 a 3 están ocupados con valores numéricos. Sin ambargo, estos bits deben leerse o interpretarse como si en realidad fuesen los bits 10, 11, 12 y 13. Si cada uno de estos bits tuviese valor 1, resultaría el número binario

0011110000000000

que corresponde al 15360 decimal.

El valor binario mínimo sería:

0000000000000000

que significa tanto como el 0 decimal. En el apéndice hemos mencionado todos los valores posibles que pueden expresarse mediante estos cuatro bits. De momento ya le decimos: si sólo el bit 0 tiene valor 1 (correspondiente al bit 10 en el cálculo), hemos creado el valor decimal 1024.

¿Qué indicamos con el bit 3 del registro 2? Con él definimos el inicio de la tabla de nombres de patrones, es decir, el lugar de la VRAM donde se memoriza la configuración de la pantalla (carácter por carácter, o sea punto por punto).

Si no asignamos ningún valor al registro 2, para cada SCREEN valdrá lo siguiente:

	inicio	longitud
SCREEN 0:	0	960
SCREEN 1:	6144	768
SCREEN 2:	6144	768
SCREEN 3:	2048	768

Puede modificar la posición de la tabla de nombres de patrones en la VRAM cambiando los bits del registro 2. De esta forma, por ejemplo, puede memorizar con SCREEN 0 varias pantallas en la VRAM y hojearlas rápidamente.

vea programa 2 del apéndice

Ejemplos de aplicación del registro 2:

VDP(2)=&B0100

El inicio de la tabla de nombres de patrones debe encontrarse en la siguiente dirección de la VRAM:

&B0001000000000000 = número decimal 5120

VDP(2)=&B1111

El inicio de la tabla de nombres de patrones debe

encontrarse en la siguiente dirección de la VRAM:

&B0011110000000000 = número decimal 15360

Registro 3

Contenido actual llamado con:
PRINT PEEK(&HF3E2)
en forma binaria (8 bits completos): 00000000
asignación con: VDP(3)=...

Aunque en este caso puedan asignarse valores a todos los 8 bits, el resultado no corresponde al real. Lo que ocurre es que el valor de los 8 bits es interpretado en un byte de 16 bits desde la posición 13 hasta la 6.

Es decir, si cada uno de los 8 bits del registro 3 contenía un 1, resultaría el número binario

0011111111000000

que corresponde al número decimal 16320.

El valor binario mínimo sería:

0000000000000000

que significa tanto como el 0 decimal. En el apéndice podrá encontar todos los valores posibles que pueden expresarse con estos 8 bits. De momento sólo mencionamos: si sólo está activado a 1 el bit 0 (correspondiente al bit 6 en el cálculo), hemos creado el valor decimal 64.

¿Qué indicamos con los 8 bits del registro 3? Con ellos definimos el inicio de la tabla de colores, es decir, el lugar de la VRAM donde se memoriza la información de color referida a gráficos/texto en la pantalla.

Si no asignamos un valor nuevo al registro 3, para los SCREENs vale lo siguiente:

	inicio	longitud
SCREEN 0:	no existente	
SCREEN 1:	8192	32
SCREEN 2:	8192	6144
SCREEN 3	0	2048

Puede cambiar de sitio la posición inicial de la tabla de colores de la VRAM modificando los bits del registro 3. De esta forma, por ejemplo, con SCREEN 1 puede memorizar en la VRAM varias informaciones diversas respecto a cada uno de los 32 bytes de color y hojearlos rápidamente.

Ejemplos de aplicación del registro 3:

VDP(3)=&B01010101

El inicio de la tabla de colores debe encontrarse en la siguiente dirección de la VRAM:

&B0001010101000000 = número decimal 5440

VDP(3)=$B11111111

El inicio de la tabla de colores debe encontrarse en la siguiente dirección de la VRAM:

&B0011111111000000 = número decimal 16320

Registro 4

Contenido actual llamado con:
PRINT PEEK(&HF3E3)
en forma binaria (8 bits completos): 00000001
asignación con: VDP(4)=...

Sólo los bits 0 a 2 están ocupados con valores numéricos. Sin embargo, estos bits deben leerse, o sea interpretarse, como si en realidad fuesen los bits 11, 12 y 13. Es decir, si estos tres bits tuviesen valor 1 resultaría el número binario

0011100000000000

que corresponde al número decimal 14336.

El valor binario mínimo sería

0000000000000000

que significa tanto como el 0 decimal. En el apéndice podrá consultar todos los valores posibles que pueden construirse con estos tres bits. De momento ya anticipamos: si sólo está activado a 1 el bit 0 (correspondiente al bit 11 en el cálculo), hemos creado el valor decimal 2048.

¿Qué indicamos con los tres bits del registro 4? Con ellos determinamos dónde se encuentra el inicio de la tabla de nombres de patrones, o sea la configuración de los caracteres o el gráfico de alta resolución.

Si no asignamos ningún nuevo valor al registro 4, para cada SCREEN valdrá lo siguiente:

	inicio	longitud
SCREEN 0:	2048	2048
SCREEN 1:	0	2048
SCREEN 2:	0	6144
SCREEN 3	no existente	

Puede cambiar de sitio la posición inicial de la tabla del generador de patrones en la VRAM, modificando los bits del registro 4. De esta forma, por ejemplo, con SCREEN 0 puede memorizar varios juegos de caracteres diferentes en la VRAM y escoger según necesidad el juego deseado.

vea programas 3 y 14 del apéndice

Ejemplos de aplicación del registro 4:

VDP(4)=&B010

El inicio de la tabla del generador de patrones debe encontrarse en la siguiente dirección de la VRAM:

&B0001000000000000 = número decimal 4096

VDP(4)=&B111

El inicio de la tabla del generador de patrones debe encontrarse en la siguiente dirección de la VRAM:

&B0011100000000000 = número decimal 14336

Registro 5

Contenido actual llamado con:
PRINT PEEK(&HF3E4)
en forma binaria (8 bits completos): 00000000
asignación con: VDP(5)=...

Sólo los bits 0 a 6 contienen valores numéricos. Sin embargo, estos bits deben leerse o interpretarse como si en realidad fuesen los bits 7 a 13. Esto significa que si estos bits estuviesen activados en 1, resultaría el número binario

0011111110000000

que corresponde al número decimal 16256.

El valor binario mínimo sería

0000000000000000

que significa tanto como el 0 decimal. En el apéndice podrá encontrar todos los valores posibles que pueden indicarse con estos 7 bits. De momento ya le anticipamos esto: si sólo se activa a 1 el bit 0 (correspondiente al bit 7 en el cálculo), habremos creado el valor decimal 128.

¿Qué indicamos con los siete bits del registro 5? Con ellos se determina el inicio de la tabla de atributos de sprites, es decir el lugar de la pantalla donde se encuentran sprites en ese momento (indicación de coordenadas X,Y), el color que tienen y el tipo de sprites a que hacen referencia en la VRAM.

Si no asignamos un nuevo valor al registro 5, para cada SCREEN vale lo siguiente:

	inicio	longitud
SCREEN 0:	no existente	
SCREEN 1:	6912	128
SCREEN 2:	6912	128
SCREEN 3	6912	128

Puede cambiar de sitio la posición inicial de la tabla de atributos de sprites en la VRAM, modificando los bits del registro 5. De esta forma, por ejemplo, con SCREEN 1 puede memorizar varias posiciones y colores de sprites en la VRAM y cambiarlas rápidamente.

Ejemplos de aplicación del registro 5:

VDP(5)=&B0101010

El inicio de la tabla de atributos de sprites debe encontrarse en la siguiente dirección de la VRAM:

&B0001010100000000 = número decimal 5376

VDP(5)=&B1111111

El inicio de la tabla de atributos de sprites debe encontrarse en la siguiente dirección de la VRAM:

&B0011111110000000 = número decimal 16256

Registro 6

Contenido actual llamado con:
PRINT PEEK(&HF3E5)
en forma binaria (8 bits completos): 00000000
asignación con: VDP(6)=...

Sólo los bits 0 a 2 contienen valores numéricos. Sin embargo, estos bits deberán ser leídos o interpretados como si en realidad fuesen los bits 11, 12 y 13. Es decir, si estos tres bits tuviesen valor 1, resultaría el número binario

0011100000000000

que corresponde al número decimal 14336.

El menor valor binario sería:

0000000000000000

que significa tanto como el 0 decimal. En el apéndice figuran todos los valores posibles que pueden indicarse con estos tres bits. De momento les anticipamos esto: si sólo el bit 0 está activado a 1 (correspondiente al bit 11 en el cálculo), habremos creado el valor decimal 1024.

¿Qué indicamos con los tres bits del registro 6? Con ellos se determina el inicio de la tabla del generador de sprites, o sea el lugar de la VRAM donde se memoriza la configuración de hasta 256 sprites del tamaño 8*8 según el comando SCREEN, es decir hasta 64 sprites del tamaño 16*16.

Si no asignamos ningún nuevo valor al registro 6, para cada SCREEN vale lo siguiente:

	inicio	longitud
SCREEN 0:	no existente	
SCREEN 1:	14336	2048
SCREEN 2:	14336	2048
SCREEN 3	14336	2048

Puede correr la posición inicial de la tabla del generador de sprites en la VRAM, modificando los bits del registro 6. De esta forma, por ejemplo, con SCREEN 1 puede memorizar en la VRAM varios grupos de hasta 256 sprites de 8*8 cada uno y hojearlos rápidamente.

Ejemplos de aplicación del registro 6:

VDP(6)=&B010

El inicio de la tabla del generador de sprites debe encontrarse en la siguiente dirección de la VRAM:

&B0001000000000000 = número decimal 4096

VDP(6)=&B111

El inicio de la tabla del generador de sprites debe encontrarse en la siguiente dirección de la VRAM:

&B0011100000000000 = número decimal 14336

Registro 7

Contenido actual llamado con:
PRINT PEEK(&HF3E6)
en forma binaria (8 bits completos): 11110100
asignación con: VDP(7)=...

Según el SCREEN escogido, los ocho bits (bits 0 a 7) tienen significados diferentes:

Si trabajamos con SCREEN 0, los bits 0 a 3 llevan la información referente al color del fondo actual en ese momento, mientras que los bits 4 a 7 contienen la información referente al color del primer plano. Como ya debería saberse, el SCREEN 0 como SCREEN único con respecto a esto sólo dispone de 2 colores que pueden escogerse como colores de fondo y de primer plano de entre los 15 diferentes matices disponibles. La forma más fácil de entrar esta información del color en el registro 7 consiste en convertir los 8 bits en 2 valores hexadecimales, puesto que: cada una de 2 cifras hexadecimales ocupa exactamente 4 bits, de manera que podemos distinguir claramente el color del fondo al lado del color del primer plano (por ejemplo, F1: color del primer plano=F=15=blanco, color del fondo=1=negro).

Si trabajamos con SCREEN 1, 2 ó 3, los bits 0 a 3 contienen la información referente al color de los bordes, mientras que los bits 4 a 7 son responsables de nuevo del color del primer plano. También en este caso parece conveniente alimentar el registro 7 directamente con cifras

hexadecimales, porque al igual que con SCREEN 0 basta aquí un número hexadecimal.

Ejemplos de aplicación del registro 7:

VDP(7)=&H11110001 = hexadecimal F1

En el caso de encontrarnos en SCREEN 0, el color del primer plano=F=15=blanco y el color del fondo=1=negro. Si nos encontramos en SCREEN 1, 2 ó 3, el color del primer plano es igualmente F=15=blanco, mientras que el del área del borde es 1=negro.

Registro 8

Tal y como ya mencionamos anteriormente, este registro no sirve para la escritura sino sólo para la lectura, en contraposición a los registros 0 a 7.

Contenido actual llamado con:
PRINT PEEK(&HF3E7)
en forma binaria (8 bits completos): 11001101
puede leerse con: PRINT VDP(8)

Bits 0 a 4: Como usted ya sabe, existen dificultades en la representación cuando se encuentran más de cuatro sprites en una horizontal de la pantalla: el quinto sprite o bien cualquier sprite adicional ya no será representado íntegramente, es decir que rige la ley siguiente : se mantienen los sprites con números inferiores, o sea los que se encuentran en la parte superior desde el punto de vista tridimensional, mientras que los sprites sobrantes con números superiores ya no serán representados (es posible que incluso con 8 sprites aún puedan distinguirse algunas partes).

Los bits 0 a 4 contienen la información sobre el número del quinto sprite (en el caso de llegar a ese extremo), que

pueda haberse sumado a los cuatro sprites ya existentes en una horizontal de la pantalla y provocar así la superposición molesta.

vea programa 5 del apéndice

El bit 5 tiene valor 1 cuando dos sprites colisionan en la pantalla y se ha activado el control de interrupción (poniendo a 1 el bit 5 del registro 1 o efectuando los comandos en BASIC ON SPRITE GOSUB y SPRITE ON). Si no hay colisión entre sprites, el valor del bit = 0. Sin embargo, debe actuar con precaución al leer el registro 8 del VDP para saber si dos sprites han colisionado, puesto que la lectura del registro 8 provoca la desconexión del control de interrupción (correspondiente al comando en BASIC SPRITE OFF). Para este fin es mejor leer la posición de memoria &HF3E7, que se va actualizando contínuamente según el contenido del registro de VDP.

El bit 6 tiene valor 1 cuando hay más que 4 sprites en una línea de la pantalla. Por ello es conveniente descifrar el número del sprite causante a través de los bits 0 a 4 del registro 8, para poder tomar medidas contra este efecto molesto lo más rápidamente posible. El bit 6 tiene valor 0 cuando en ese momento no se encuentren más de cuatro sprites en cada horizontal (referida a puntos).

Finalmente, el bit 7 trabaja indirectamente junto con todos los comandos del VDP: este bit se activa al valor 1 unas 50 veces por segundo, cada vez que se termina una configuración de la pantalla. Esto se cumplirá contínuamente para los programadores en BASIC dada la lentitud de este lenguaje de programación. Sin embargo, este bit puede proporcionar explicaciones importantes para los programadores en lenguaje máquina.

Ejemplos de aplicación del registro 8:

```
PRINT BIN$(VDP(8))
```

en BASIC corresponde a: PRINT BIN$(PEEK(&HF3E7))

resultado: 11111111

Bits 0 a 4: en ese momento existe un sprite (número binario 11111=31 decimal) que se ha sumado a los cuatro sprites ya existentes en una horizontal.

Bit 5: el 1 indica además, que se ha producido una colisión entre dos sprites en la pantalla.

Bit 6: este 1 confirma la afirmación hecha por los bits 0 y 4: más de cuatro sprites se encuentran en una horizontal (referida a puntos).

Bit 7: la pantalla fue renovada como siempre cada 1/50 segundos, por lo que este bit es = 1.

Comandos   Gráficos   BASE

Variable dimensionada que informa sobre la configuración de la VRAM

vea también los programas del apéndice: BASE 1
BASE 2
BASE 3
BASE 4
BASE 5
BASE 6
BASE 7
BASE 8
BASE 9
BASE 10
BASE 11
BASE 12
BASE 13
BASE 14
BASE 15
programa 14

Para poder aprovechar correctamente este comando usted ya debe tener ciertos conocimientos sobre el almacenamiento de la RAM de visualización del sistema MSX. Bien es cierto que el comando BASE no confecciona ningún gráfico de texto directamente, pero en contrapartida le indica exactamente la configuración que la memoria RAM tiene en cada momento con el correspondiente SCREEN. Además puede utilizar el comando BASE para crear su propia configuración de la memoria RAM de visualización.

En realidad, BASE es más bien una variable que un comando. Según el valor asignado a BASE, la configuración de la VRAM se verá afectada directa o indirectamente.

Empecemos a investigar el comando BASE (o mejor dicho: la variable BASE) con sus diversos contenidos. La variable BASE consiste en un campo dimensionado de 20 valores que ya está provisto de números al conectar el ordenador MSX. Entre el siguiente programa para cerciorarse de ello:

vea listado de BASE 1 del apéndice

Resultado:

0 = 0	1 = 0
2 = 2048	3 = 0
4 = 0	5 = 6144
6 = 8192	7 = 0
8 = 6912	9 = 14336
10 = 6144	11 = 8192
12 = 0	13 = 6912
14 = 14336	15 = 2048
16 = 0	17 = 0
18 = 6912	19 = 14336

Interesante, ¿pero a qué viene esto? Vayamos por orden: estos 20 valores numéricos no pueden ser utilizados juntamente, sino que representan más bien cuatro grupos de cinco números cada uno. Estos cuatro grupos de cinco números están asignados a las diversas representaciones de pantalla a los que podemos dirigirnos con el sistema MSX y el chip gráfico incorporado:

valores 0 a 4: SCREEN 0
valores 5 a 9: SCREEN 1
valores 10 a 14: SCREEN 2
valores 15 a 19: SCREEN 3

Entremos en más detalles: Los cinco valores numéricos hacen referencia respectivamente a cinco direcciones iniciales o base de la memoria RAM de visualización, que son:

valor 0: aquí se almacena la configuración de la pantalla, referida siempre a 256 caracteres diferentes

valor 1: memoriza el color de los caracteres representados (no en SCREEN 0 y 3)

valor 2: almacena punto por punto el aspecto de cada uno de los caracteres o sea de los gráficos

valor 3: aquí se memoriza todo lo relacionado con

el color y la posición de los sprites (no en SCREEN 0)

valor 4: aquí se encuentra la parte de la memoria donde se guarda el aspecto de los sprites (no en SCREEN 0)

Los valores 5 a 9 contienen la información correspondiente para SCREEN 1, los valores 10 a 14 contienen la información para SCREEN 2 y los valores 15 a 19 finalmente la información para SCREEN 3.

Tal como puede verse fácilmente en la lista anterior, en SCREEN 0 y SCREEN 3 sólo pueden utilizarse razonablemente algunas partes de la RAM de visualización según la lista de BASE. Realicemos pues los ejemplos en SCREEN 1:

1) La dirección inicial de nuestro contenido de pantalla según la tabla BASE se halla en la dirección 6144. Para comprobarlo borramos la pantalla (con SCREEN 1) y POKEamos el número 2 a la dirección 6144:

```
SCREEN 1
VPOKE 6144,2
```

Debe prestar atención a no entrar el comando POKE por equivocación. Tenga en cuenta que aquí trabajamos con la memoria RAM de visualización y el comando que sirve para modificar estas posiciones de memoria es el VPOKE (Video-POKE) y no el POKE.

¿Qué ha sucedido? De hecho no ha sucedido casi nada, pero no deje de observar la esquina superior izquierda de la pantalla, porque podría ser que no pueda examinar esta esquina. En este caso tendrá que ajustar su televisor o monitor de tal manera que permita distinguir la esquina del extremo superior izquierdo. Ahí debería representarse una cara 'smiley'. Si miramos en la tabla ASCII (vea apéndice) reconoceremos esta cara risueña en el carácter ASCII 2. Es decir que con el comando VPOKE 6144,2 le hemos indicado a nuestro ordenador MSX que dibuje una cara risueña en la

esquina superior izquierda de la pantalla. Para que podamos verla todos, pongamos esta cara en el centro de la primera línea de pantalla. Al utilizar SCREEN 1 disponemos de 32 caracteres por línea, o sea que entramos otra cara en la posición de memoria 6144+15, que corresponde a la posición central:

```
VPOKE 6144 + 15,2
```

Partiendo de esta cara aislada queremos representar ahora toda una columna de ellas en la pantalla, de arriba a abajo:

vea listado de BASE 2 del apéndice

Como puede ver, su ordenador MSX no se inmuta si hace aparecer el listón de las teclas de función en la pantalla (protegiéndolas así de una posible reescritura o sea que ya sólo dispone de 23 líneas de trabajo en lugar de 24), cuando esté POKEando en la memoria VRAM. De golpe aparece también un cara 'smiley' en la línea 24 que podrá volver a borrarse entrando otro VPOKE, a menos que vuelva a definir de nuevo el listón de las teclas de función o borre la pantalla utilizando CLS o SCREEN 1 (también puede borrar la cara del listón de las teclas de función entrando CLS o pulsando la tecla SHIFT).

Podemos aprovechar este segundo margen de pantalla (ya sabemos que el verdadero margen se modifica con ayuda del tercer parámetro del comando COLOR), manteniendo con WIDTH nuestro campo de pantalla lo más reducido posible, conservando el listón de teclas de función. Ejecute el siguiente ejemplo y modifíquelo según sus propias ideas:

vea listado de BASE 3 del apéndice

Su pantalla está rodeada ahora con caras risueñas a la izquierda, derecha y abajo. Esta margen de la pantalla abierto hacia arriba se mantiene hasta que se realice la entrada de CLS o de un comando SCREEN. También puede volver a borrar el segundo margen de la pantalla variando la

anchura de la misma o bien POKEando de nuevo en sus correspondientes posiciones.

Del programa Base 3 puede sacarse la conclusión de que en SCREEN 1 la pantalla llega de la posición de memoria 6144 (BASE(5)) hasta la 6911, que son en total 768 posiciones de memoria, puesto que 32 caracteres por línea * 24 (número de líneas) dan 768.

Comparado con eso, ¿qué ocurre en SCREEN 0? BASE(0) nos indica el principio de la pantalla:

PRINT BASE (0)

resultado: 0

Aquí disponemos de 24 líneas de 40 caracteres cada una. Esto significa que en SCREEN 0 la memoria de pantalla tiene una longitud de 960 caracteres, desde la dirección 0 hasta la 959.

Observemos ahora las informaciones sobre las dos pantallas gráficas (SCREEN 2 y SCREEN 3), en lo concerniente a su configuración (no su asignación de color ni los sprites):

para SCREEN 2: PRINT BASE (10)

resultado: 6144

para SCREEN 3: PRINT BASE (15)

resultado: 2048

Antes de ocuparnos del color en las pantallas gráficas y de texto, presentamos aquí los datos técnicos de la pantalla SCREEN 3 de baja resolución: la representación (vea variable BASE(15)) comienza en la posición de memoria 2048 y tiene igualmente una longitud de 768 posiciones.

A continuación se muestra un cuadro resumen de los cuatro

SCREENs en lo que se refiere a la representación de imagen:

SCREEN 0, comienzo 0, longitud 960, variable BASE(0)
SCREEN 1, comienzo 6144, longitud 768, variable BASE(5)
SCREEN 2, comienzo 6144, longitud 768, variable BASE(10)
SCREEN 3, comienzo 2048, longitud 768, variable BASE(15)

Observemos ahora la segunda parte de las direcciones de BASE, donde se memoriza el color de letras y gráficos:

2) La variable BASE(1) nos indica el valor 0. Si se ha dado cuenta, esto no puede ser verdad, puesto que BASE(0) ya tenía el valor 0. En realidad, esta posición sólo tiene valor 0 porque en SCREEN 0 no podemos indicar el color de forma diferenciada si no utilizamos el comando COLOR. Este comando es válido para toda la pantalla de golpe, al igual que en SCREEN 1, y sólo requiere dos posiciones de memoria. Por esta razón no se crea ninguna 'paleta de colores' especial para SCREEN 0 en la VRAM.

Vayamos directamente a SCREEN 1:

```
10 SCREEN 1
20 PRINT BASE (6)
```

resultado: 8192

La 'paleta de colores' comienza en la dirección de memoria 8192 de la VRAM, y tiene una longitud de sólo 32 caracteres que además están repartidos locamente para su aplicación. Antes de profundizar en ello, entre el siguiente ejemplo:

vea listado de BASE 4 del apéndice

Al ejecutar este programa, la pantalla se convierte en algo confuso e incluso ilegible según la cantidad y diversidad de los caracteres representados. Para ver la razón de ello, ponga de nuevo el color en su estado inicial (entrando SCREEN 1; aquí ya no basta realizar un simple cambio de color utilizando COLOR) y añada las siguientes líneas de

programa:

<u>vea listado de BASE 4a del apéndice</u>

Ahora ejecuta el programa completo con RUN. Secuencias de ocho caracteres (por ejemplo, de H a O ó bien de 0 a 7) agrupados han modificado sus colores de fondo y de primer plano. Por esta razón, al pulsar las teclas 7 y 8 vemos colores muy diferentes que al pulsar las teclas 6 y 7 (los dos últimos caracteres numéricos pertenecen al mismo grupo, mientras que el 8 ha tomado el color de otro grupo de 8 caracteres).

¿Pero cómo está organizada la entrada de 32 bytes? Cambiemos la posición de memoria 8198, que contiene la asignación del color para los caracteres numéricos 0 a 7:

VPOKE 8198,255

resultado: fondo blanco, carácter blanco

VPOKE 8198,0

resultado: fondo transparente, carácter transparente

VPOKE 8198,31

resultado: fondo blanco, carácter negro

Para verlo con más claridad, observemos los valores numéricos entrados en su formato binario (sólo con 0 y 1):

255 = 11111111
0 = 00000000
31 = 00011111

Debemos repartirnos las ocho cifras en dos grupos de cuatro. Entonces el número decimal 255 tiene el significado binario 1111 y 1111, que traducido nuevamente al sistema decimal

corresponde a 15 y 15. El primer número (los cuatro 1 de la izquierda) indica el color del primer plano; el segundo número define la naturaleza del fondo del carácter (en este caso también es blanco). En nuestro tercer ejemplo (31 o en binario 0001 1111) corresponde pues al primer plano (el propio carácter) negro (en binario 0001 = 1 decimal), mientras que el fondo aparece de color blanco (en binario 1111 = 15 decimal) en la pantalla.

Para acabar de entender todo esto le mostramos dos ejemplos más, invirtiendo el orden (después podrá comenzar con las pruebas):

color de primer plano 13, color de fondo 7
binario 1101 0111; decimal 215
VPOKE 8198,215

color de primer plano 7, color de fondo 13
binario 0111 1101; decimal 125
VPOKE 8198,125

Continuemos ahora con la asignación de color en SCREEN 2. El valor de BASE (11) nos indica aquí que la memoria de color del gráfico de alta resolución comienza en la posición 8192. En la VRAM de color se almacenan 6144 caracteres, fácilmente demostrable con un simple cálculo: 192 caracteres pueden tomar cualquier color en cada secuencia de caracteres; puesto que existen 32 caracteres por línea, la multiplicación de 32 por 192 nos da el resultado final de 6144 ya conocido.

Investiguemos ahora más detenidamente esta variedad de colores con ayuda de un programa ejemplo:

vea listado de BASE 5 del apéndice

Una pantalla de varios colores nos muestra de esta forma la configuración de la memoria VRAM.

Pero a decir verdad no podemos distinguirlo todo, ya que en cada mancha de color de 8*1 puntos (como se nos muestran en forma de minúsculas barras horizontales en la pantalla) se memorizan siempre dos colores, un color del primer plano y otro del fondo. El almacenamiento de los colores del primer plano y del fondo de un carácter se realiza de la misma manera que se explicó en SCREEN 1: los cuatro primeros bits para el color del primer plano y los cuatro últimos para el color del fondo. A partir de aquí podrán calcularse los números hexadecimales que facilitan la entrada, como por ejemplo, los colores del primer plano 7 y del fondo FF dan en forma binaria 0111 1111 o en la decimal 127.

Finalmente vayamos a la asignación de color en SCREEN 3: el gráfico de baja resolución está repartido en 64*48 puntos de pantalla. Otra vez se almacenan dos cosas en un byte: el color del primer plano y el del fondo. Esto significa que podemos efectuar otro cálculo para conocer las posiciones de memoria ocupadas por esta pantalla de colores: (64/2)*48 = 1536 caracteres.

Por razones de mayor claridad le mostramos otra vez un resumen de los cuatro SCREENs en cuanto respecta a su representación en color:

SCREEN 0, comienzo /, longitud /, variable BASE(1)
SCREEN 1, comienzo 8192, longitud 32, variable BASE(6)
SCREEN 2, comienzo 8192, longit. 6144, variable BASE(11)
SCREEN 3, comienzo 0, longitud 1536, variable BASE(16)

Inspeccionemos ahora la tercera parte de las direcciones BASE. Aquí se anota punto por punto el aspecto de cada carácter en la pantalla:

3) Puesto que en SCREEN 0 no disponemos de representación gráfica, esta memorización es asignada al aspecto de cada carácter. La memorización comienza por la dirección 2048 y se extiende por 2048 posiciones de memoria. ¿Cómo y cuándo se almacena ahora el aspecto de un carácter?

Con respecto a ello debemos recordar que el ordenador interpreta cada carácter como un número; de esta forma el ordenador concibe el espacio como CHR$(32), las comillas como CHR$(34) y la letra A finalmente como CHR$(65) (vea también en la explicación de comandos respecto a ASC y CHR$()).

Veamos de qué forma almacena nuestro ordenador MSX los caracteres en la memoria:

<u>vea listado de BASE 6 del apéndice</u>

```
resultado: 32
           80
           136
           136
           248
           136
           136
           0
```

Sin embargo, esto no basta todavía para representar la letra (en nuestro caso la A), puesto que tras cada número decimal se esconde la clave de encendido/apagado, o sea 1/0 para el ordenador:

<u>vea listado de BASE 7 del apéndice</u>

```
resultado: 00100000  =    1
           01010000  =   1 1
           10001000  =  1   1
           10001000  =  1   1
           11111000  =  11111
           10001000  =  1   1
           10001000  =  1   1
           00000000  =
```

Se preguntará seguramente por qué las tres columnas de la derecha de la matriz de 8*8 se han dejado libres, o sea indefinidas. Hay dos razones para ello: por un lado debe quedar al menos una columna libre para garantizar el límite hacia el carácter que se une por la derecha. Las otras dos columnas libres hacen referencia a la característica del gráfico de alta resolución del ordenador MSX, ya que se pretenden visualizar 40 caracteres por línea. Puesto que la pantalla de texto SCREEN 0 también es producida por el procesador gráfico, la representación se reduce a una matriz de 6*8 por visualizar 40 caracteres/línea, ya que 6 * 40 = 240, mientras que 8 * 40 = 320; sin embargo 'sólo' disponemos de 256 puntos por línea.

Además hay que contestar la pregunta sobre el por qué se deja la última línea (línea 7) de la letra A libre. Como ya sabe, el juego de caracteres del MSX se compone de letras minúsculas y mayúsculas. Las letras minúsculas tales como j y g requieren una longitud inferior. Para indicarla también en la pantalla, con la mayoría de caracteres se deja la última línea libre para diferenciarlo.

Usted puede modificar el aspecto de la matriz de los caracteres de cualquier forma. Para tener en cuenta este fin, hemos incluído aquí un llamado generador de caracteres:

vea programa 14 del apéndice

Ya ve que la realización de su creatividad no encuentra límites.

En SCREEN 1 el trato con el aspecto de caracteres almacenado en la memoria es idéntico al explicado anteriormente en SCREEN 0, con la diferencia de que en ese caso la dirección inicial de la VRAM es la 0. La longitud de esta parte de la memoria también es de 2048 bytes.

Puesto que en SCREEN 1 los caracteres están ligeramente estirados (ya no son 40 caracteres como en SCREEN 0, sino tan sólo 32 caracteres/línea), ahora también se indicará el aspecto completo de los caracteres (matriz de 8*8, ya que 8 * 32 da 256, que es precisamente la resolución de nuestro ordenador MSX). Es evidente que lo más normal sería desarrollar un juego de caracteres especialmente para SCREEN 1. Esto es precisamente lo que hemos hecho y usted puede empezar -si no lo ha hecho ya- a teclear el juego de caracteres representado con ayuda del generador de caracteres.

Hablemos ahora de las pantallas gráficas: en nuestras anteriores reflexiones básicas, en lo tocante a los gráficos, nos hicimos la pregunta: ¿cuántas posiciones de memoria son precisas para representar 256 * 192 puntos? Hasta ahora sólo conocemos el comienzo de la memorización. En SCREEN 2 habíamos entrado antes un pequeño programa para demostrar la representación de colores. Puesto que el gráfico de puntos de alta resolución se visualiza sólo con ayuda de colores, debemos producir ahora color y gráfico de alta resolución en este programa. Entonces podrá verse claramente que en la definición del color existe tanto un color de primer plano como uno de fondo. Con el fin de demostrarlo, entre el siguiente programa, por favor:

vea listado de BASE 8 del apéndice

En primer lugar, en la línea 20 se crea un patrón punto por punto (en cada byte se definen ocho puntos -piense en punto activado y desactivado o 1, 0-) en la pantalla y a continuación se vierte el color encima del mismo (cada ocho puntos en la horizontal aceptan 'sólo' dos colores diferentes).

Si desea practicar esta resolución también con el gráfico de alta resolución en SCREEN 3, debo decepcionarle desgraciadamente, puesto que en SCREEN 3 esta lista no está ocupada según el comando BASE (mejor dicho: BASE(17)). Recordará que el gráfico de 64 * 48 puntos ya fue destapado

juntamente con los 16 colores por la lista de colores.

Resumimos de nuevo lo que ocurre en SCREEN 0 a 3 en lo que se refiere a gráficos de alta resolución:

SCREEN 0, comienzo 2048, longit. 2048, variable BASE(2)
SCREEN 1, comienzo 0, longitud 2048, variable BASE(7)
SCREEN 2, comienzo 0, longitud 6144, variable BASE(12)
SCREEN 3, comienzo /, longitud /, variable BASE(17)

4) Finalmente llegamos a los sprites y su almacenamiento en la RAM de vídeo. Por suerte su ordenador MSX dispone de gran número de comandos BASIC que nos facilitan considerablemente el trabajo con estos cuerpos de movimiento fluído. La cuarta dirección BASE hace referencia a los atributos de los sprites en cada SCREEN.

Si empezamos por SCREEN 0, ya podemos pasar de entrada: aquí no existen sprites y, por lo tanto, no hay tampoco lista de atributos.

En SCREEN 1 ya cambia la cosa: la lista de atributos (en los ejemplos expuestos a continuación verá su significado) comienza en dirección 6912 de la VRAM y tiene una longitud de 128 bytes.

Ahora profundizaremos un poco en el contenido de estos 128 bytes, pudiendo aplicarlo igualmente para los demás SCREENs, puesto que el trato de los sprites y la correspondiente lista de atributos son idénticas en todos ellos.

En cada SCREEN (exceptuando SCREEN 0) tenemos la posibilidad de visualizar 32 sprites en la pantalla. Sin embargo, esto sólo es posible si se cumple una medida de precaución: en un nivel horizontal no pueden representarse nunca más de cuatro sprites. En caso de no cumplir esta condición sucederán cosas muy extrañas en la pantalla: un quinto sprite adicional borra el sprite de mayor número. Sin embargo, una vez superada la fase de deslizamiento, se volverán a mostrar los cuatro 'viejos' sprites como si nada hubiera ocurrido.

La lista de atributos contiene cuatro informaciones referentes a cada uno de los 32 sprites (4 * 32 = 128 bytes):

El primer carácter indica la coordenada Y.
El segundo carácter indica la coordenada X.
El tercer carácter indica el sprite (vea el comando SPRITE$ y el apartado 5 del presente capítulo) a representar.
El cuarto carácter indica el color del sprite y además el lugar donde se encuentra el punto de salida para X,Y.

Antes de proceder a la aplicación hay que interpretar el cuarto carácter con mayor exactitud. Para ello debemos observarlo más detenidamente en su representación binaria: observemos, por ejemplo, qué ocurre cuando el sprite es blanco (color 15):

15 decimal = 00001111 binario

Los bits primero, segundo, tercero y cuarto contienen el código del color. Comenzando por la derecha, los bits quinto, sexto y séptimo de la expresión binaria no tienen ningún significado. ¿Pero qué ocurre con el bit número ocho? Para saberlo hemos de profundizar un poco más en la materia del proceso de números de los ordenadores:

Ya hemos comprobado que los dos primeros caracteres de los respectivos atributos del sprite corresponden a las coordenadas Y y X del sprite en la pantalla. Como usted ya sabe, la pantalla tiene una resolución de 256*192 puntos. No hay ningún problema, dirá usted, podemos hacer deslizar nuestros sprites punto por punto en la pantalla. Esto es verdad si observa tan sólo la vertical; sin embargo, el problema se presenta al observar la horizontal más detenidamente: el ordenador MSX representa el sprite en la pantalla considerando su esquina superior izquierda como la coordenada base. No hay ningún problema si hacemos deslizar

el sprite de derecha a izquierda:

vea listado de BASE 9 del apéndice

Funciona de maravilla: el sprite sale de la pantalla deslizándose prácticamente y después no será visto más.

Al experimentar lo mismo en el sentido contrario, inmediatamente nos llevamos una sorpresa desagradable:

vea listado de BASE 10 del apéndice

Realmente es muy bonita la forma en que el sprite se introduce lentamente en la pantalla por la derecha, pero al alcanzar la posición 0,0 ... el sprite no sale de la pantalla por la izquierda sino que permanece quieto en la esquina superior esperando más instrucciones.

Ahora podemos entrar en acción activando a 1 el octavo bit de nuestro carácter (sumamos 128 a nuestra concesión de color). Esto indica al ordenador el estado de emergencia en que nos hallamos en este momento y las coordenadas serán interpretadas ahora de la forma siguiente:

X=X-32   Y=Y

Su ordenador MSX pone simplemente a -32 sus coordenadas de cálculo para la visualización del sprite, con lo que éste desaparece inmediatamente de la pantalla.

El ordenador MSX ejecuta este procedimiento complicado automáticamente en el BASIC al alimentarlo con valores negativos. Pero tenga cuidado, que demasiado alimento le hace perder el apetito. Puede continuar el bucle FOR ... NEXT (línea 40) en el campo negativo, pero cuando llegue demasiado lejos, el sprite vuelve a aparecer súbitamente por el lado derecho de la pantalla (sin deslizarse lentamente). Puede evitar este efecto imprevisto si no rebasa el campo normal en más de 32 cifras (de -32 a +287). Sin embargo, queremos observar por puro interés lo complicado que este

procedimiento resulta para el ordenador en la VRAM:

vea listado de BASE 11 del apéndice

Hasta aquí nuestro programa referente a la representación de derecha a izquierda. Pero ahora queremos conseguir el efecto de entrada deslizándose por la derecha:

vea listado de BASE 11a del apéndice

Además de la concesión de atributos para los sprites (color, posición en la pantalla, número del sprite en la correspondiente tabla del generador), también es preciso poder definir el aspecto de un sprite. Ello se memoriza en el quinto apartado de su correspondiente parte de la RAM de vídeo:

5) El BASIC MSX permite definir el aspecto de un máximo de 32 sprites y emitirlos simultáneamente en la pantalla. ¿Dónde y cómo se almacena esta información en la memoria?

Como ya comprobamos al hablar de los atributos de los sprites, SCREEN 0 no dispone de sprites. Pero podemos comenzar a definirlos a partir de SCREEN 1. Puesto que la memorización de los sprites se realiza de igual forma para SCREEN 1, 2 y 3, en este capítulo profundizaré tan sólo en la configuración de la memoria de SCREEN 1 como muestra orientativa para el trabajo con SCREEN 2 y 3.

El almacenamiento de los sprites comienza por la dirección 14336 de la RAM de vídeo y tiene una longitud de 2048 posiciones. En cada caso podemos definir más sprites de los que puedan representarse simultáneamente en la pantalla: atendiendo las limitaciones (no más de cuatro sprites al mismo tiempo en un nivel horizontal, para evitar que el quinto cubra partes del sprite de mayor número), pueden representarse 32 sprites simultáneamente en la pantalla.

Si queremos mostrar sólo sprites pequeños (matriz de 8*8), accesibles a través de SCREEN ,0 o SCREEN ,1, en la RAM de

vídeo pueden definirse los aspectos de hasta 256 sprites, lo que corresponde a la memoria disponible que resulta del cálculo 256*8 = 2048).

Puesto que el nivel de memorización para el aspecto de los sprites comienza en la posición de memoria 14336, el sprite 0 se encuentra en las posiciones 14336 a 14343, el sprite 1 en las posiciones 14344 a 14351, y así sucesivamente hasta llegar al sprite 255 en las posiciones 16377 a 16384. Para facilitar su ilustración, el aspecto de los sprites puede entrarse en forma binaria, aunque es aconsejable convertirlo seguidamente en el sistema decimal para ahorrar memoria:

binario	00000000	decimal	0
	01111110		126
	01111110		126
	01100110		102
	01100110		102
	01111110		126
	01111110		126
	00000000		0

A continuación puede emitirse el sprite en la pantalla utilizando el comando PUT SPRITE:

vea listado de BASE 12 del apéndice

El sprite aparece ahora en el centro de la pantalla; compárelo con nuestro diseño realizado.

Si queremos evitarnos el paso intermedio (conversión binario-decimal), también podemos procesar los ceros y unos directamente. El correspondiente programa sería el siguiente:

vea listado de BASE 13 del apéndice

Puesto que en este capítulo pretendemos orientarnos en BASE sólo con ayuda de la configuración de la memoria, sustituyendo para ello el amplio juego de comandos del BASIC

MSX (especialmente en lo referente a los gráficos) por accesos directos a la memoria, también simularemos el comando gráfico PUT SPRITE de las líneas 70 ó 90 por un acceso directo. ¿Recuerda todavía el procedimiento a seguir?

Correcto, únicamente hemos de almacenar en la memoria los atributos del sprite en cuestión. Recordemos: los atributos incluyen:

1) la posición Y (en este caso = 96)
2) la posición X (en este caso = 128)
3) el número del sprite (en este caso = 0)
4) el color del sprite, así como el posible factor de desplazamiento (esto último no es necesario en nuestro caso, ya que el sprite se sitúa en el centro de la pantalla), o sea color = blanco = 15, por ejemplo.

Como que la primera entrada ya se refiere al primer sprite (dirección 6912 de la lista de atributos) podemos sustituir las líneas 70 ó 90 directamente por:

```
70 o bien 90 VPOKE 6912,96
71 o bien 91 VPOKE 6913,128
72 o bien 92 VPOKE 6914,0
73 o bien 93 VPOKE 6915,15
```

¿Pero cómo debemos proceder para representar sprites de mayor tamaño? En este caso identificamos nuestra intención con el parámetro 2 ó 3 (ampliado) en el segundo parámetro, por ejemplo, del comando SCREEN ,2 o SCREEN ,3.

La memoria RAM de vídeo puede almacenar 256 sprites del tamaño 8*8; pero para definir cada uno de los sprites de 16*16 se precisan cuatro del tamaño 8*8 para completar el sprite "mayor", es decir que en la memoria sólo podemos disponer de 64 sprites diferentes preparados para ser llamados (64*32=2048, que corresponde a la memoria disponible para almacenar sprites). A pesar de ello es posible emitir simultáneamente 32 sprites del tamaño 16*16

en la pantalla.

El almacenamiento de los sprites en la RAM de vídeo se realiza de la siguiente manera: primeramente se lee la parte izquierda superior, después la izquierda inferior y finalmente las partes derecha superior e inferior. Cada parte tiene un tamaño de 8*8 puntos. Dibujemos ahora nuestro patrón anterior en el tamaño 16*16, calculando a la vez los correspondientes valores decimales:

sprite 1:	00000000	00000000	sprite 3	0	0
	01111111	11111110		127	254
	01111111	11111110		127	254
	01100000	00000110		96	6
	01100000	00000110		96	6
	01100000	00000110		96	6
	01100011	11000110		99	19
	01100010	01000110		98	70
sprite 2:	01100010	01000110	sprite 4	98	70
	01100011	11000110		99	19
	01100000	00000110		96	6
	01100000	00000110		96	6
	01100000	00000110		96	6
	01111111	11111110		127	254
	01111111	11111110		127	254
	00000000	00000000		0	0

Realmente ha salido bastante extenso y laborioso. Miremos la forma y el lugar donde podemos leer nuestros valores decimales en un programa:

vea listado de BASE 14 del apéndice

Veamos ahora la forma de colocar nuestro sprite de 16*16 en la memoria sin necesidad de hallar su valores decimales y sin hacer uso de los comandos gráficos en BASIC MSX:

vea listado de BASE 15 del apéndice

Seguramente sabrá apreciar ahora la importancia que tiene, al confeccionar y utilizar sprites, disponer por un lado de un BASIC lujoso (los comandos PUT SPRITE y sobre todo la asignación de variable SPRITE$ resultan especialmente útiles), y por otro además del direccionamiento directo a un editor de sprites, que nos evita realmente mucho trabajo ... y esa es precisamente la tarea de un ordenador: ejecutar trabajos molestos de forma rápida y sin grandes esfuerzos y dejar para el hombre aquellas tareas que requieren reflexión, lógica y creatividad (de forma parecida que una calculadora de bolsillo). Sin embargo, al igual que la utilización de una calculadora de bolsillo no debería hacernos olvidar el cálculo mental, el trabajo con el ordenador tampoco debería hacerlo: a pesar de todo lujo es importante comprender todo lo que sucede en el interior del ordenador, en este caso particularmente en su RAM de vídeo, puesto que nosotros queremos controlar y supervisar las máquinas, pero no ser controlados por ellas.

Con el siguiente resumen finalizamos este capítulo:

El almacenamiento de los sprites, su cantidad, su comienzo en la memoria, su posición en la misma y finalmente la correspondiente variable BASE:

SCREEN 0:
no sprites, BASE(4)

SCREEN 1,0 y SCREEN 1,1:
256 sprites de 8*8, comienzo: 14336, longitud: 2048, BASE(9)

SCREEN 1,2 y SCREEN 1,3:
64 sprites de 16*16, comienzo: 14336, longitud: 2048, BASE(9)

SCREEN 2,0 y SCREEN 2,1:
256 sprites de 8*8, comienzo: 14336, longitud: 2048, BASE(14)

SCREEN 2,2 y SCREEN 2,3:
64 sprites de 16*16, comienzo: 14336, longitud: 2048, BASE(14)

SCREEN 3,0 y SCREEN 3,1:
256 sprites de 8*8, comienzo: 14336, longitud: 2048, BASE(19)

SCREEN 3,2 y SCREEN 3,3:
64 sprites de 16*16, comienzo 14336, longitud: 2048, BASE(19)

BASE 1

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=0 TO 19
40 PRINT N"=";BASE(N),
50 NEXT N
```

BASE 2

```
10 COLOR 1,15
20 SCREEN 1
30 FOR N=6144+15 TO 6144+751 STEP 32
40 VPOKE N,2
50 NEXT N
```

BASE 3

```
10 COLOR 1,15
20 SCREEN 1
30 KEY ON
40 WIDTH 20
50 COLOR 15,4,5
60 FOR N=6144 TO 6911
70 VPOKE N,2
80 PRINT STRING$(20,"A");
90 NEXT N
```

BASE 4

```
10 COLOR 1,15
20 SCREEN 1
30 FOR N=8192 TO 8192+32
40 VPOKE N,INT(RND(1)*255)
50 NEXT N
```

BASE 4a

```
21 FOR N=32 TO 255
22 PRINT CHR$(N);
23 NEXT N
```

BASE 5

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=8192 TO 8192+6144
40 VPOKE N,INT(RND(1)*255)
50 NEXT N
```

BASE 6

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=2048+(65*8) TO 2048+(65*8)+7
40 PRINT VPEEK(N)
50 NEXT N
```

BASE 7

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=2048+(65*8) TO 2048+(65*8)+7
40 B$=BIN$(VPEEK(N))
50 PRINT STRING$(8-LEN(B$),"0")+B$
60 NEXT N
```

BASE 8

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=8192 TO 8192+6143
40 VPOKE N,INT(RND(1)*255)
50 VPOKE N-8192,INT(RND(1)*255)
60 NEXT N
```

BASE 9

```
10 COLOR 1,15
20 SCREEN 2,3
30 SPRITE$(0)=STRING$(32,"U")
40 FOR N=0 TO 255
50 FOR M=0 TO 10
60 NEXT M
70 PUT SPRITE 0,(N,0)
80 NEXT N
90 GOTO 90
```

BASE 10

```
10 COLOR 1,15
20 SCREEN 2,3
30 SPRITE$(0)=STRING$(32,"U")
40 FOR N=255 TO 0 STEP -1
50 FOR M=0 TO 10
60 NEXT M
70 PUT SPRITE 0,(N,0)
80 NEXT N
90 GOTO 90
```

BASE 11

```
10 COLOR 1,15
20 SCREEN 2,3
30 SPRITE$(0)=STRING$(32,"U")
40 FOR N=0 TO 255
50 VPOKE 6912,0
60 VPOKE 6913,N
70 VPOKE 6914,1
80 VPOKE 6915,1
90 NEXT N
100 GOTO 100
```

BASE 11a

```
10 COLOR 1,15
20 SCREEN 2,3
30 SPRITE$(0)=STRING$(32,"U")
40 FOR N=255 TO 0 STEP -1
50 VPOKE 6912,0
60 VPOKE 6913,N
70 VPOKE 6914,1
80 VPOKE 6915,1
90 NEXT N
100 GOTO 100
```

BASE 12

```
10 COLOR 1,15
20 SCREEN 1,0
30 FOR N=0 TO 7
40 READ A
50 VPOKE 14336+N,A
60 NEXT N
70 PUT SPRITE 0,(128,96)
80 GOTO 80
90 DATA 0,126,126,102,102,126,126,0
```

BASE 13

```
10 COLOR 1,15
20 SCREEN 1,0
30 FOR N=0 TO 7
40 READ A$
50 A$="&B"+A$
60 A=VAL(A$)
70 VPOKE 14336+N,A
80 NEXT N
90 PUT SPRITE 0,(128,96)
100 GOTO 100
110 DATA 00000000
120 DATA 01111110
130 DATA 01111110
140 DATA 01100110
150 DATA 01100110
160 DATA 01111110
170 DATA 01111110
180 DATA 00000000
```

BASE 14

```
10 COLOR 1,15
20 SCREEN 1,2
30 FOR N=0 TO 3
40 FOR M=0 TO 7
50 READ A
60 VPOKE 14336+(N*8)+M,A
70 NEXT M
80 NEXT N
90 VPOKE 6912,96
100 VPOKE 6913,128
110 VPOKE 6914,1
120 VPOKE 6915,1
130 GOTO 130
140 DATA 0,127,127,96,96,96,99,98
150 DATA 98,99,96,96,96,127,127,0
160 DATA 0,254,254,6,6,6,198,70
170 DATA 70,198,6,6,6,254,254,0
```

BASE 15

```
10 COLOR 1,15
20 SCREEN 1,2
30 FOR N=0 TO 15
40 READ A$
50 B$="&B"+LEFT$(A$,8)
60 VPOKE 14336+N,VAL(B$)
70 NEXT N
80 RESTORE
90 FOR N=0 TO 15
100 READ A$
110 B$="&B"+RIGHT$(A$,8)
120 VPOKE 14352+N,VAL(B$)
130 NEXT N
140 VPOKE 6912,96
150 VPOKE 6913,128
160 VPOKE 6914,1
170 VPOKE 6915,1
180 GOTO 180
190 DATA 0000000000000000
200 DATA 0111111111111110
210 DATA 0111111111111110
220 DATA 0110000000000110
230 DATA 0110000000000110
240 DATA 0110000000000110
250 DATA 0110001111000110
260 DATA 0110001001000110
270 DATA 0110001001000110
280 DATA 0110001111000110
290 DATA 0110000000000110
300 DATA 0110000000000110
310 DATA 0110000000000110
320 DATA 0111111111111110
330 DATA 0111111111111110
340 DATA 0000000000000000
```

Comandos Gráficos VPEEK

Lee el contenido de posiciones de la memoria RAM de vídeo (VRAM)

vea también los programas del apéndice: VPEEK 1
VPEEK 2
VPEEK 3
VPEEK 4
VPEEK 5
VPEEK 6
VPEEK 7
programa 12

En contraposición a muchos otros ordenadores, en los equipos de MSX no sólo diferenciamos entre RAM y ROM, sino además entre RAM y VRAM. Explicando brevemente estos términos diremos: En la ROM se encuentra el BASIC y el sistema operativo del ordenador MSX. Esta memoria llamada fija, comprende 32 kilobytes en los ordenadores MSX y se compone principalmente de rutinas en lenguaje máquina enfiladas, que proporcionan la ejecución de los comandos en BASIC MSX. La RAM es la memoria de lectura/escritura, cuyo tamaño difiere en los diversos ordenadores MSX. En la RAM no sólo se almacenan los programas, sino que aquí nuestro ordenador MSX también memoriza cosas que no nos parecen tan evidentes, como un listado de programa: las variables, la ocupación de las teclas de función, los actuales colores de primer plano, fondo y área de los bordes, etc.

Además de estos dos bloques de memoria, los ordenadores MSX tienen otra memoria separada de lectura/escritura, que se encarga exclusivamente de los gráficos. En ella se almacena el aspecto de los caracteres, la representación completa tanto en la pantalla gráfica como en la de texto, el aspecto de los sprites y su actual posicionamiento. Aquí se sitúa no sólo la representación gráfica por puntos, sino además la asignación del color a los patrones. La memoria VRAM tiene un tamaño de 16 KB en los ordenadores MSX.

Mientras que el comando PEEK nos permite leer de la RAM o la

ROM, el comando VPEEK sirve para la correspondiente lectura de la VRAM. Puesto que la VRAM tiene un tamaño de 16 KB, el comando VPEEK sólo podrá leer el contenido de las posiciones 0 a 16383 de la memoria. Al consultar números superiores o inferiores, en la pantalla aparece el mensaje de error: 'Illegal function call'.

En este capítulo no pretendemos ocuparnos en profundidad de los posicionamientos exactos en la VRAM. Al tratar de los comandos BASE ya entramos detenidamente en el tema. En este capítulo más bien queremos aclarar, con ayuda de un ejemplo, cómo y dónde se memoriza en la VRAM el aspecto de los caracteres en SCREEN 0, y la forma en que podemos efectuar la correspondiente lectura en las posiciones de la memoria VRAM utilizando VPEEK:

vea listado de VPEEK 1 del apéndice

Realmente parece un poco abstracto si ahora entra las siguientes 8 cifras para definir el aspecto de la letra 'A':

32, 80, 136, 136, 248, 136, 136, 0

Pero debemos profundizar más para reconocer realmente la forma de nuestra 'A' tras estas cifras. Para ello hemos de concienciarnos de que nuestro ordenador MSX de hecho no entiende más que los estados '0' y '1'. Sin embargo, puesto que los cálculos con estos dos estados se efectúan a gran velocidad en el cerebro del ordenador (hasta más de 100000 veces por segundo en los ordenadores MSX), que además reconoce consecuentemente determinados estados de encendido/apagado o 1/0 como razonables, se le convertirá pues en astuto mediante la rapidez y ya no se apreciará su presunta necedad.

Con ayuda del comando BIN$ podemos convertir los números previamente en esta representación 0/1:

vea listado de VPEEK 2 del apéndice

Con un poco de fantasía podrá reconocer ahora nuestra 'A' punto por punto. Aclaremos esto un poquito más, anteponiendo a las líneas de salida incompletas la cantidad adecuada de ceros (puesto que cada número decimal entre 0 y 255 puede representarse mediante un número binario de ocho cifras 0 ó 1):

vea listado de VPEEK 3 del apéndice

Si ahora prescinde de los ceros podrá distinguir punto por punto la configuración exacta de nuestra 'A'. ¿Cómo dice? ¿Más claro todavía?

vea listado de VPEEK 4 del apéndice

Consulte en el apéndice la forma en que hemos imprimido el juego de caracteres completo de su ordenador con el programa 12. El procedimiento seguido en él es el mismo que el utilizado en el programa VPEEK que acabamos de ver:

vea programa 12 del apéndice

De forma parecida podemos observar el sprite punto por punto o bien toda la representación gráfica de la memoria VRAM en la pantalla. Sin embargo, no se imagine que la lectura de las posiciones de la memoria VRAM en SCREEN 2 sea muy fácil, porque su MSX es un ordenador de color y gráficos y colores, que se muestran como un conjunto en la pantalla, ocupan dos zonas diferentes de la memoria VRAM. De ellas resultará entonces el propio gráfico de color. Es conveniente que prosiga con sus investigaciones en la VRAM con nuestros programas VPEEK 1 a VPEEK 4, donde se memorizan los aspectos de sprites y caracteres (en la descripción de los comandos BASE encontrará la repartición de la VRAM).

Finalmente queremos descubrirle un truco: ¿cómo puede representarse una entrada diez veces mediante la lectura de la VRAM? Entre el siguiente programa para verlo:

vea listado de VPEEK 5 del apéndice

Ahora leemos en la pantalla y escribimos lo leído en diversas posiciones de la misma.

vea listado de VPEEK 6 del apéndice

VPEEK nos sirve incluso para leer en la pantalla la lista de las teclas de función y hacerlas aparecer de nuevo en otro lugar:

vea listado de VPEEK 7 del apéndice

VPEEK 1

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 PRINT VPEEK(2048+65*8+N-1);
50 NEXT N
```

VPEEK 2

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 PRINT BIN$(VPEEK(2048+65*8+N-1))
50 NEXT N
```

VPEEK 3

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 PRINT STRING$(8-LEN(BIN$(VPEEK(2048+6
5*8+N-1))),"0")+BIN$(VPEEK(2048+65*8+N-1
))
50 NEXT N
```

VPEEK 4

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 A$=STRING$(8-LEN(BIN$(VPEEK(2048+65*8
+N-1))),"0")+BIN$(VPEEK(2048+65*8+N-1))
50 FOR M=1 TO 8
60 IF MID$(A$,M,1)="1" THEN PRINT CHR$(2
19);ELSE PRINT " ";
70 NEXT M
80 PRINT
90 NEXT N
```

VPEEK 5

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 A$="Ordenador MSX"
50 PRINT A$
60 FOR M=1 TO 10
70 FOR N=1 TO LEN(A$)
80 VPOKE M*20+N,VPEEK(N-1)
90 NEXT N
100 NEXT M
110 LOCATE 0,10
```

VPEEK 6

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 A$="Ordenador MSX"
50 PRINT A$
60 FOR M=1 TO 10
70 FOR N=1 TO LEN(A$)
80 VPOKE INT(RND(1)*940+20),VPEEK(N-1)
90 NEXT N
100 NEXT M
110 LOCATE 0,10
```

VPEEK 7

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 FOR M=1 TO 10
50 FOR N=1 TO 40
60 VPOKE M*40+N,VPEEK(23*40+N-1)
70 NEXT N
80 NEXT M
```

Comandos Gráficos VPOKE
Escribe valores numéricos en las posiciones de la VRAM

vea también los programas del apéndice: VPOKE 1
VPOKE 2
VPOKE 3
VPOKE 4
VPOKE 5
VPOKE 6
VPOKE 7
programa 2
programa 3
gráfico leng. máqu.

Al igual que cualquier otro ordenador, también su MSX dispone de dos memorias diferentes: por un lado la ROM, que es la memoria sólo de lectura (Read Only Memory) y que contiene entre otros el BASIC, así como también la definición base del aspecto de los caracteres y la ocupación original de las teclas de función. Por otro lado existe la RAM (Random Access Memory), que memoriza toda la información entrada por usted, sean contenidos de variables o líneas de programa, hasta que vuelva a desconectar el ordenador. Es decir, la RAM es una memoria de libre escritura y lectura, pero la información sólo se mantiene en ella mientras el ordenador disponga de flujo eléctrico (por esta razón existen memorias masivas tales como cassettes y diskettes, que almacenan esta información 'fugaz' de forma permanente).

Si deseamos leer en la RAM o la ROM, nos servimos del comando en BASIC PEEK. Además indicamos la correspondiente posición de memoria (en un Homecomputer de 8 bits tal como su ordenador MSX, las posiciones de memoria se extienden como máximo de 0 a 65535) y provocamos la emisión en pantalla mediante un PRINT precedente.

Si por el contrario desea escribir algo en la memoria, debe utilizar el comando POKE (por ello también se habla de 'POKEar en la memoria'); sin embargo, como ya mencionamos al diferenciar ROM y RAM: sólo podrá escribir en la memoria de

escritura, o sea la RAM; POKEar en la zona ROM no tendrá ningún efecto.

A diferencia de la mayoría de ordenadores domésticos, los ordenadores MSX presentan una particularidad respecto a la RAM: por un lado disponen de memoria RAM libre para recoger variables y programas; por otro lado, sin embargo, disponen además de otra zona de memoria completamente separada de la RAM, que se encarga exclusivamente de las cosas relacionadas con la representación en pantalla. Esta memoria abarca 16 Kilobytes en los ordenadores MSX (más de 16000 posiciones de memoria) y contiene información (según elección del SCREEN) sobre la representación de los caracteres en la pantalla, el aspecto de los caracteres emitidos punto por punto, información sobre la configuración de hasta 256 sprites diferentes, su actual posición, el gráfico de color punto por punto, etc.

Puesto que esta memoria RAM está relacionada con la configuración del vídeo de pantalla, su denominación no es simplemente RAM sino VRAM (Vídeo-RAM). También puede POKEar directamente en la VRAM todas las actividades ejecutables con cualquier comando gráfico, lo cual exige sin embargo un conocimiento detallado de la configuración de la VRAM, que podrá adquirir sucesivamente en el capítulo referente al comando BASE.

Acabo de hablar de POKEar en la VRAM. A causa de su separación de la memoria RAM normal, será imposible escribir en ella utilizando los comandos PEEK y POKE,sino que precisa de otros especiales de MSX VPEEK (o sea, para la emisión en pantalla: PRINT VPEEK(N)) y VPOKE (o sea, para la entrada VPOKE N,M). VPEEK lee la información en la VRAM, mientras que VPOKE permite la escritura o almacenamiento de información en ella.

Observemos más detalladamente el trabajo con VPEEK y VPOKE con ayuda de dos pequeños ejemplos:

vea listado de VPOKE 1 del apéndice

Este programa no pretende explicar la configuracion de la VRAM, sino aclarar el significado del comando VPEEK: en primer lugar ha escrito el logo del MSX en la parte superior izquierda de pantalla. Aquí presuponemos que ya sabe que la configuración de la pantalla en SCREEN 0 comienza por la posición 0 de la memoria VRAM (vea en BASE). Durante la ejecución del programa observamos detenidamente los tres primeros caracteres de la VRAM:

```
PRINT VPEEK(0): 77 = M
PRINT VPEEK(1): 83 = S
PRINT VPEEK(2): 88 = X
```

Las tres posiciones de memoria llevan realmente la información sobre las tres letras 'MSX' que entramos anteriormente en la primera línea del programa con PRINT "MSX". Para demostrarlo, las POKEamos simplemente en un segundo programa con VPOKE de nuevo hacia las tres primeras posiciones de la VRAM,después de limpiar la pantalla:

vea listado de VPOKE 2 del apéndice

Se preguntará seguramente cómo se consiguen los valores numéricos '77', '83' y '88' para las letras 'M', 'S' y 'X'. Ello está relacionado con que los caracteres representados en la pantalla son codificados en forma de valores numéricos, por ejemplo, 32 en lugar del espacio (consulta también los comandos ASC y CHR$ de este libro). Esto significa que debemos entrar los códigos ASC de cada carácter para poder emplazar letras en la pantalla mediante VPOKE. Observémoslo en el programa siguiente, que coloca de esta forma, carácter por carácter sucesivamente, una frase indicada por usted en la esquina superior izquierda de la pantalla de SCREEN 0:

vea listado de VPOKE 3 del apéndice

Ahora se preguntará seguramente: ¿por qué tanta complicación, si se puede conseguir el mismo resultado

efectuando PRINT "..."? Queremos hacerle la presentación detallada de VPOKE por tres razones:

1) El comando PRINT trabaja en sus comandos aislados de lenguaje máquina de la misma forma complicada que acabamos de presentar parcialmente. Conociendo el comando VPOKE más detalladamente, le permite echar una ojeada más profunda en la complejidad de los pequeños pasos realizados en lenguaje máquina, que en realidad trabaja con pasos mucho más pequeños, siendo el direccionamiento directo a la VRAM su verdadero objetivo.

2) Hablamos aquí del comando VPOKE, no sólo para que usted lo conozca en la teoría, sino para que además lo comprenda en la práctica.

3) No hay ningún comando BASIC que le facilite la lectura en la pantalla de texto (a diferencia del comando POINT de la pantalla gráfica). Si desea hacerlo, deberá acostumbrarse al menos al comando VPEEK, por muy poco confortable que parezca.

Antes de POKEar ahora desenfrenadamente en la memoria VRAM, le recordamos que actúe con precaución: en primer lugar conozca la configuración de la VRAM más a fondo a través del comando BASE, de lo contrario pueden ocurrir desastres por culpa de algún VPOKE efectuado arbitrariamente:

vea listado de VPOKE 4 del apéndice

Numerosas letras han empezado a multiplicarse rápidamente, ya no puede leerse casi nada y se siente la tentación de desconectar el ordenador y volver a conectarlo. Sin embargo, el problema puede resolverse más fácilmente: borre la pantalla, o sea la memoria VRAM, con un comando SCREEN, que devuelve la VRAM a su estado original y todo se ha resuelto.

Sin embargo, queremos estudiar (aunque con cuidado) una segunda aplicación del comando VPOKE visible directamente en SCREEN 0: queremos modificar el aspecto de un carácter -en este caso el espacio- en la pantalla, de tal forma que parezca un pequeño tablero de ajedrez; es decir, la pantalla se compone de prácticamente de 1000 casillas de 64 cuadros cada una (de hecho sólo son 48 cuadros, puesto que en SCREEN 0 se representan 'sólo' 3/4 de los caracteres; vea SCREEN):

<u>vea listado de VPOKE 5 del apéndice</u>

¿Qué ha ocurrido? En SCREEN 0 el aspecto del espacio comienza en la posición 2048+32*8 de la memoria VRAM y continúa hasta llegar a la 2048+32*8+7. ¿Por qué 32*8? Tal como ya señalé anteriormente, el espacio corresponde al código ASC 32. La dirección 2048 de VRAM coincide con el comienzo de la zona de la VRAM, que se encarga del aspecto de cualquier carácter representable. Esto significa que debemos saltarnos las primeras 32*8 posiciones de memoria (32*8 porque la cuenta empieza por el carácter 0) para llegar a nuestro carácter 32. A continuación se pondrá 85 y 170 en éste y los 7 valores siguientes. La razón del por qué precisamente estos números, se explica con la decodificación mediante el comando BIN$, que es la representación numérica más pequeña para nuestro ordenador MSX (aquí no hay 10 estados válidos como en el sistema decimal, sino tan sólo 2(encendido y apagado, o sea 0 y 1)):

```
BIN$(85)= 01010101
BIN$(170)=10101010
```

Ahora se han activado puntos de pantalla en todas las posiciones del 'espacio' que contienen un 1. Por el contrario, en las posiciones que contienen un 0 binario no se activan los puntos de pantalla.

No se preocupe, por supuesto podrá devolver el 'espacio' fácilmente a su estado original:

1) entrando cualquier comando SCREEN N

2) ejecutando de nuevo el programa anterior tras modificar las líneas siguientes (POKEamos sencillamente un 0 en las posiciones de pantalla antes modificadas):

```
40 VPOKE 2048+32*8+N,0
50 VPOKE 2048+32*8+N+1,0
```

Ya he señalado antes que el POKEar es muy importante para aquellos que pueden trabajar en lenguaje máquina, con lo cual no utilizan comando BASIC tales como PRINT. Para demostrarle que VPOKE no es útil para la representación de pantalla en BASIC, ejecute el siguiente test de velocidad:

vea listado de VPOKE 6 del apéndice

El resultado de este pequeño programa debería coincidir en todos los ordenadores MSX:

1- Escribir en SCREEN 0 con PRINT:
3.36 segundos

2- Escribir en SCREEN 0 con VPOKE:
3.64 segundos

3- Escribir en SCREEN 0 con PRINT STRING$:
0.58 segundos

4- Escribir en SCREEN 0 con rutina en lenguaje máquina:
0.02 segundos

¡Vaya diferencia! Profundizaré más en ello en el capítulo referente a la programación gráfica en lenguaje máquina. De momento debe bastar este test para demostrar la velocidad del lenguaje máquina, así como la lentitud de ejecución de los comandos VPOKE.

Finalizaremos dando algunas pequeñas advertencias:

1) En cuanto conozca a fondo la VRAM, podrá trabajar con VPEEK y VPOKE en 16 Kilobytes de posiciones de memoria, que corresponden exactamente a las posiciones 0 a 16383. La entrada detrás del comando VPOKE no puede ser inferior a 0 ni superior a 255. Si la posición de memoria referida en VPOKE o el valor a colocar en aquella posición exceden de los límites admitidos, se produce el mensaje de error 'Illegal function call'.

2) Puesto que en SCREEN 0 no aprovecha la totalidad de los 16 KB de VRAM para la representación de pantalla, es posible utilizar la restante memoria disponible con la debida precaución como memoria de variables o de pantalla ampliada. Con respecto a ello consulte el apéndice o el comando BASE para saber en qué lugar del SCREEN escogido por usted dispone de VRAM libre. Sin embargo, debe tener precaución especialmente porque el comando SCREEN borra toda la memoria VRAM, y con ello también sus datos de forma irreversible.

3) Se puede conseguir un efecto elegante con VPOKE, configurando la memoria VRAM a la que no puede dirigirse directamente a través del BASIC como margen: de esta forma convierte, por ejemplo, las teclas de función y los bordes derecho e izquierdo que puedan existir (con WIDTH más pequeños que el valor extremo) en marcos, tal como muestra el siguiente ejemplo:

vea listado de VPOKE 7 del apéndice

Pero guárdese de entrar a continuación CLS, un comando SCREEN, otro WIDTH o KEY OFF o bien KEY ON: inmediatamente se volvería a destruir al menos parcialmente el margen de la pantalla.

4) Aunque conozca la configuración de la VRAM, se encontrará con grandes dificultades al crear, por ejemplo, una línea en modo gráfico de alta resolución utilizando sencillamente VPOKE. Ello se debe por un lado a que su ordenador MSX gasta

muy poca memoria (parecido a la representación con 0/1 de 8 puntos, utilizando una representación binaria resultante de convertir un número decimal de la zona 0 a 255, al definir el aspecto de caracteres tal como fue descrito anteriormente). Además, la representación gráfica y el color se almacenan en dos partes de la VRAM separadas entre sí. No es tan fácil relacionar ambas zonas adecuadamente.

VPOKE 1

```
10 SCREEN 0
20 WIDTH 40
30 COLOR 1,15
40 PRINT "MSX"
50 PRINT
60 FOR N=0 TO 2
70 PRINT VPEEK(N);
80 NEXT N
```

VPOKE 2

```
10 SCREEN 0
20 WIDTH 40
30 COLOR 1,15
40 VPOKE 0,77
50 VPOKE 1,83
60 VPOKE 2,88
70 LOCATE 0,1
```

VPOKE 3

```
10 SCREEN 0
20 WIDTH 40
30 COLOR 1,15
40 PRINT
50 PRINT
60 INPUT "Escriba una frase corta:
          ";A$
70 PRINT
80 PRINT "Esta frase se POKEa ahora a la
         VRAM a partir de la posicion 0
"
90 FOR N=1 TO 1000
100 NEXT N
110 FOR N=1 TO LEN(A$)
120 VPOKE N-1,ASC(MID$(A$,N,1))
130 NEXT N
```

VPOKE 4

```
10 SCREEN 0
20 WIDTH 40
30 COLOR 1,15
40 FOR N=1 TO 1000
50 VPOKE 2048+INT(RND(1)*2048),INT(RND(1
)*255)
60 NEXT N
70 LIST
```

VPOKE 5

```
10 SCREEN 0
20 COLOR 1,15
30 FOR N=0 TO 7 STEP 2
40 VPOKE 2048+32*8+N,170
50 VPOKE 2048+32*8+N+1,85
60 NEXT N
```

VPOKE 6

```
10 CLEAR 200,50000!
20 KEY OFF
30 SCREEN 0
40 WIDTH 40
50 COLOR 1,15
60 TIME=0
70 FOR N=1 TO 960
80 PRINT "*";
90 NEXT N
100 A=TIME
110 CLS
120 TIME=0
130 FOR N=0 TO 959
140 VPOKE N,42
150 NEXT N
160 B=TIME
170 CLS
180 TIME=0
190 FOR N=1 TO 24
200 PRINT STRING$(40,"*");
210 NEXT N
220 C=TIME
230 CLS
240 FOR N=1 TO 12
250 READ A$
260 POKE 49999!+N,VAL("&H"+A$)
270 NEXT N
280 DEFUSR=50000!
290 TIME=0
300 D=USR(0)
```

```
310 D=TIME
320 DATA 3e,2a,21,00,00,01,c0,03,cd,56,0
0,c9
330 CLS
340 PRINT "Segundos que ha tardado:"
350 PRINT
360 PRINT "con PRINT:";:PRINT USING "##.
##";A/50
370 PRINT "con VPOKE:";:PRINT USING "##.
##";B/50
380 PRINT "con PRINT STRING$ :";:PRINT U
SING "##.##";C/50
390 PRINT "con codigo maquina:";:PRINT U
SING "##.##";D/50
```

VPOKE 7

```
10 SCREEN 0
20 KEY ON
30 COLOR 1,15
40 WIDTH 34
50 FOR N=0 TO 967
60 VPOKE N,ASC("*")
70 NEXT N
80 LOCATE 0,23
90 FOR N=1 TO 24
100 PRINT
110 NEXT N
```

Comandos Gráficos CLS

Borra la representación en pantalla de diversos SCREENs

vea también los programas del apéndice: CLS 1

COLOR 4

Con ayuda de este comando BASIC puede borrarse la representación en pantalla de cualquier SCREEN. Sólo se conserva la indicación de las teclas de función (si ha sido activado previamente con KEY ON) así como el cursor (si se ha mantenido en su estado normal, o bien provocado anteriormente su visualización en la pantalla con LOCATE ,,1), mientras que se borra cualquier otra representación de la pantalla.

Es cierto que también puede limpiarse la pantalla utilizando el comando SCREEN, pero entonces suceden más cosas. Ello puede causarnos problemas involuntariamente, puesto que toda la RAM de vídeo es devuelta a su estado original; es decir, que cualquier carácter redefinido anteriormente volverá a su forma habitual, los grupos de ocho caracteres activados con diversos colores en SCREEN 1 (vea BASE) vuelven a mostrarse en sus originales colores de fondo o de primer plano, etc.

Así que a primera vista sucede lo mismo, tanto si utilizamos el comando SCREEN como el CLS para borrar la pantalla, pero hay que tener precaución.

Otra posibilidad de limpiar la pantalla consiste en entrar líneas vacías con ayuda del comando PRINT (sólo válido para las pantallas de texto). Sin embargo, en este caso es conveniente recurrir al comando CLS (CLEAR SCREEN) porque trabaja a mayor velocidad.

Mientras que la instrucción COLOR en SCREEN 0 y 1 provoca una reacción inmediata en cuanto respecta a la modificación del color de fondo (segundo parámetro), la entrada aislada de COLOR no basta para cambiar el color en las pantallas gráficas. No podremos modificar el color de fondo hasta que

utilicemos el comando CLS:

vea listado de CLS 1 del apéndice

Por cierto:

1) Si queremos aplicar lo mismo a los demás SCREENs, se considera: CLS repinta la pantalla con el color de fondo indicado en último lugar (segundo parámetro del comando COLOR).

2) No podemos producir el mismo efecto en modo directo, si en lugar del comando CLS pulsamos una vez la tecla CLS/HM o entramos PRINT CHR$(12) en el modo programa.

vea listado de COLOR 4 del apéndice

Con cada cambio de SCREEN se efectúa automáticamente un CLS. Podemos evitarlo (cambio de SCREEN sin borrado de pantalla), llamando directamente las rutinas de la ROM:

```
SCREEN 0: &H0078
SCREEN 1: &H007B
SCREEN 2: &H007E
SCREEN 3: &H0081
```

Aunque esto parezca todavía algo confuso, entre las siguientes instrucciones para su ejecución directa:

```
SCREEN 0
DEFUSR=&H007E
A=USR(0)
```

Ahora puede entrar directamente desde el teclado el texto que aparece codificado en forma de gráfico en la pantalla. Por supuesto también podrá borrar la pantalla mediante CLS o volver a la pantalla de texto normal efectuando la entrada de SCREEN 0. Sin embargo, al ejecutar la instrucción SCREEN volverá a borrarse la pantalla. Realícelo sin borrarla, es decir, llamando la rutina en lenguaje máquina

correspondiente a SCREEN 0 y entre en modo directo:

```
DEFUSR=&H0078
A=USR(0)
```

Es cierto que ahora estamos de nuevo en el estado inicial, pero usted puede distinguir exactamente (ahora también legible) todo lo que antes ha tecleado 'a ciegas'.

```
CLS 1
     10 SCREEN 2
     20 COLOR 15,1
     30 DRAW "BM 0,0"
     40 OPEN "GRP:" FOR OUTPUT AS #1
     50 PRINT #1,"   Solo cambia el color
             del primer plano
     60 FOR N=1 TO 1000
     70 NEXT N
     80 CLS
     90 PRINT #1,"   Gracias a CLS ahora tamb
     ien      cambia el color del fondo"
     100 FOR N=1 TO 1000
     110 NEXT N
     120 COLOR 1,4
```

Comandos Gráficos KEY ON/OFF

Valida e invalida la indicación de las teclas de función

vea también los programas del apéndice: KEY ON 1
KEY ON 2
KEY ON 3

Al conectar nuestro ordenador MSX nos encontramos en el modo SCREEN 0, una de las pantallas más puras de texto. Cada línea contiene 37 caracteres (podemos ampliar a 40 caracteres por línea utilizando WIDTH) y podemos escribir directamente en 23 de las 24 líneas de pantalla a través del teclado. Si deseamos escribir en la línea 24, la pantalla efectúa automáticamente un scroll ascendente en una línea, puesto que la indicación de las teclas de función está integrada en el margen de la pantalla y no permite reescribir en ella.
Si precisamos más espacio en la pantalla, también podemos hacer que la indicación de las teclas de función (línea 24 de la pantalla) desaparezca. Para este fin disponemos del comando KEY OFF del MSX. Si deseamos volver a mostrar dicha indicación, entramos KEY ON y todo vuelve a su estado habitual:

KEY ON = 23 líneas de texto
KEY OFF = 24 líneas de texto

Mientras que en caso normal se indica la función de las teclas de F1 a F5, la representación varía al pulsar simultáneamente una de las dos teclas SHIFT (entonces se indica la función de las teclas F6 a F10).

Sin embargo, esta lista no es completa ni mucho menos, puesto que pueden ponerse hasta 16 caracteres en cada tecla. Si graduamos SCREEN 0 a su mayor anchura de pantalla posible (WIDTH 40), en la lista de las teclas de función se indican sólo los caracteres 1 a 7 de cada una de ellas:

1234567	1234567	1234567	1234567	1234567
F1	F2	F3	F4	F5

Sí ahora reducimos la amplitud de pantalla a 20 caracteres, la indicación de las teclas de función disminuye hasta una expresión mínima de 3 caracteres:

123	123	123	123	123
F1	F2	F3	F4	F5

Es cierto que esto hace mal efecto, pero no puede resumirse de otra forma (para la indicación completa de la ocupación de las teclas de función, su ordenador MSX dispone del comando KEY LIST).

SCREEN 0/40 caracteres: indicación de 7 caracteres
SCREEN 0/37 caracteres: indicación de 6 caracteres
SCREEN 0/20 caracteres: indicación de 3 caracteres
SCREEN 1/32 caracteres: indicación de 5 caracteres
SCREEN 1/29 caracteres: indicación de 5 caracteres

Desgraciadamente no disponemos de indicación de las teclas de función en las pantallas gráficas. Aunque sea un poco complicado y, sobre todo, requiera mucho tiempo, también en este caso podemos obtener dicha indicación. Para ello debemos forzar la salida de la memoria de las teclas de función (cinco caracteres cada uno, puesto que nos encontramos en un modo de representación de 32 y no de 40 caracteres) en la línea 24 (memorizada de &HF87F a &HF91E)

Tecla de función 1: &HF87F a &HF88E
Tecla de función 2: &HF88F a &HF89E
Tecla de función 3: &HF89F a &HF8AE
Tecla de función 4: &HF8AF a &HF8BE
Tecla de función 5: &HF8BF a &HF8CE
Tecla de función 6: &HF8CF a &HF8DE
Tecla de función 7: &HF8DF a &HF8EE
Tecla de función 8: &HF8EF a &HF8FE
Tecla de función 9: &HF8FF a &HF90E
Tecla de función 10: &HF90F a &HF91E

y consultar al mismo tiempo la lectura de la tecla SHIFT pulsada o no pulsada, de manera que se indiquen realmente las correspondientes teclas de función de 0 a 5 ó de 6 a 10. En este momento no quiero profundizar más en el comando (INP) aquí utilizado, puesto que para ello existen otros libros. Aquí tan sólo pretendemos aprovechar de forma práctica la lectura de la tecla SHIFT con ayuda de INP:

vea listado de KEY ON 1 del apéndice

Otras cuatro advertencias:

1) En cualquier momento puede leer el contenido de la actual ocupación de las teclas de función (al igual que en el programa anterior) en las posiciones de memoria &HF87F a &HF91E.

vea listado de KEY ON 2 del apéndice

2) La ocupación estándar de las teclas de función se encuentra en las posiciones &H13A7 a &H1448 de la memoria ROM del MSX. Si desea obtener esta ocupación, debe entrar directamente (o en el programa) los dos comandos siguientes en su ordenador MSX:

```
DEFUSR=&H003E
A=USR(0)
```

A continuación pulse una tecla SHIFT.

3) Si trabaja en SCREEN 0 o SCREEN 1 y la emisión de las teclas de función está validada (estado inicial o KEY ON), únicamente VPOKE le permitirá cambiar la línea 24, como se muestra en el siguiente ejemplo:

vea listado de KEY ON 3 del apéndice

El estado original (con indicación normal de las teclas de función) podrá ser recuperado fácilmente con ayuda de CLS, un comando SCREEN, pulsación de las teclas CLS/HOME o SHIFT o el comando KEY ON.

4) El cambio de la indicación de las teclas de función también se muestra durante la ejecución de un programa pulsando la tecla SHIFT, aunque sólo en el momento en que se efectúe una entrada:

```
A$=INKEY$
INPUT A
A$=INPUT$(1)
```

KEY ON 1

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "B M 12,96"
50 PRINT #1,"Teclas de funcion en SCREEN
 2"
60 PRINT #1,"        cambiando con SHIFT"
70 Z=0
80 FOR N=10 TO 210 STEP 50
90 DRAW "bm =n;,183"
100 Z=Z+1
110 A$=""
120 FOR M=1 TO 5
130 A$=A$+CHR$(PEEK(Z*16+&HF86F+M-1+(ZZ*
80)))
140 NEXT M
150 PRINT #1,A$;
160 NEXT N
170 A=INP(&HAA) AND &HFO
180 OUT &HAA,A OR 6
190 B=INP(&HA9)
200 IF B AND 1 THEN IF ZZ=1 THEN ZZ=0:LI
NE(10,183)-(255,191),15,BF
210 IF NOT B AND 1 THEN IF ZZ=0 THEN ZZ=
1:LINE(10,183)-(255,191),15,BF
220 GOTO 70
```

KEY ON 2

```
10 SCREEN 0
20 COLOR 1,15
30 A$=STRING$(16,32)
40 FOR N=5033 TO 5192 STEP 16
50 FOR M=1 TO 16
60 IF PEEK(N-1+M)>31 THEN MID$(A$,M,1)=C
HR$(PEEK(N-1+M)) ELSE MID$(A$,M,1)=" "
70 NEXT M
80 PRINT A$
90 NEXT N
```

KEY ON 3

```
10 KEY ON
20 SCREEN 0
30 COLOR 1,15
40 FOR N=920 TO 959
50 VPOKE N,INT(RND(1)*255)
60 NEXT N
```

Comandos Gráficos WIDTH

Gradúa la anchura de las pantallas de texto

vea también los programas del apéndice: WIDTH 1
programa 2

Al conectar el ordenador MSX y después de desaparecer el logo del MSX, nos encontramos en SCREEN 0, graduado en una anchura de 37 caracteres por línea de pantalla, que dispone de una indicación de las teclas de función.

A diferencia de la mayoría de ordenadores domésticos, que precisan un contínuo POKEado en la memoria, su ordenador MSX le ofrece un comando BASIC especial para graduar la anchura de la pantalla (visualización de N caracteres por línea):

WIDTH N

Sin embargo, en lugar de N no puede poner cualquier valor. Hay que respetar los siguientes límites extremos:

SCREEN 0: WIDTH 1 a 40
SCREEN 1: WIDTH 1 a 32
SCREEN 2 y 3: comando WIDTH no aplicable, o bien la entrada WIDTH se refiere automáticamente a SCREEN 0 o SCREEN 1.

¿Cómo diferenciamos las pantallas de SCREEN 0 y SCREEN 1 en la transmisión de texto? Para ello debe saber, que los ordenadores MSX disponen de un gráfico de alta resolución de 256*192 puntos. Teniendo en cuenta esto, en SCREEN 0 debe atenerse a las siguientes limitaciones:

SCREEN 0: 256/40 = 6 puntos en la matriz horizontal

mientras que en SCREEN 1 (hasta 32 caracteres/línea) puede verse la matriz de caracteres completa:

SCREEN 1: 256/32 = 8 puntos en la matriz horizontal

Esto significa que en SCREEN 0 no se visualiza el carácter completo (8*8), sino sólo los 6 primeros puntos de la lista horizontal de 8 puntos. Para evitar su efecto molesto, los fabricantes del MSX han construído su juego de caracteres de tal forma, que se ocupan sólo los 6 primeros puntos. Sin embargo, tenga siempre en cuenta esta limitación de SCREEN 0 antes de estirarse de los cabellos tras una definición conseguida de caracteres gráficos que se visualiza de forma incompleta.

En SCREEN 1 la imagen es un poco estirada, pero en contrapartida puede partir siempre de una completa visualización de los caracteres (8*8). Por esta razón hemos trabajado con SCREEN 1 en lugar de SCREEN 0 para nuestro generador de caracteres (vea programa 14 del apéndice). Si se dirige a SCREEN 1 poco después de conectar su ordenador, observará que en su estado inicial tampoco se representa el número máximo de 32, sino tan sólo 29 caracteres por línea (corresponde a WIDTH 29).

¿Por qué sólo se visualizan 37 caracteres en el estado inicial de SCREEN 0 y sólo 29 en SCREEN 1? Cada televisor, o sea cada sistema utilizado por los televisores internacionales (PAL, NTSC, SECAM) cumple normas diferentes en lo que se refiere a la representación en la pantalla. Su ordenador MSX lleva incorporado un modulador (traductor de imagen) común en España, interpretado para la norma PAL. De forma parecida se procede en América con un modulador NTSC; puesto que el MSX es un ordenador internacional, en cada país se procede según las normas correspondientes. Desgraciadamente, con ello no se eliminan las diferencias en la representación en pantalla del ordenador. De esta forma, las circunferencias dibujadas con el ordenador MSX aparecen en forma de elipse en nuestro sistema PAL y ... en representaciones de 40 caracteres en SCREEN 0 ó de 32

caracteres en SCREEN 1 ya no se distinguen absolutamente todos los caracteres (o al menos algunos se ven sólo parcialmente en el margen de la pantalla).

Para evitar estas diferencias internacionales, cada ordenador MSX crea sólo una parte de su anchura de línea completa al conectarlo. Por supuesto, usted puede cambiar con WIDTH a 40 caracteres/línea en SCREEN 0 ó bien 32 caracteres/línea en SCREEN 1, pero vea usted mismo.

Puesto que un monitor conectado, en cambio, puede graduarse tanto como desee (un locutor de televisión ya no tendría el aspecto esperado tras semejante acción de ajuste), le permite regular la imagen como quiera. Por esta razón, los monitores disponen de reguladores de ajuste que llevan inscripciones importantes para la resolución de nuestro problema, tales como: Horizontal Width o bien Vertical Width. Si usted es propietario afortunado de un monitor de este tipo, no debería tener problemas al ajustar la pantalla adecuadamente, lo que afecta tanto a la visualización completa en la pantalla como a la representación de circunferencias realmente circulares de su ordenador MSX.

Con ayuda de un corto programa quiero mostrarle otra vez que, a diferencia de SCREEN 1, en SCREEN 0 se visualiza realmente sólo una matriz de 6*8 del carácter de 8*8:

vea listado de WIDTH 1 del apéndice

El comando WIDTH cambia también la lista de las teclas de función al efectuar la entrada de una nueva anchura de pantalla. Mientras que con la graduación normal, después de la conexión (SCREEN 0, WIDTH 37) todavía puede verse la palabra COLOR de la tecla de función 1 (COLOR 15,4,4), después de cambiar la graduación a WIDTH 10 ya sólo queda la C como ayuda orientativa.

En este caso, suele ser conveniente invalidar simplemente la indicación de las teclas de función con SCREEN ,,0, ganando de esta forma incluso la línea número 24.

También puede cambiar la anchura de la pantalla con ayuda de un POKE hacia las posiciones de memoria &HF3AE, &HF3AF y &HF3B0, que hacen referencia a las anchuras de pantalla de SCREEN 0, a la de SCREEN 1 y la del SCREEN actual en ese momento (la entrada en las posiciones &HF3AE y &HF3AF no muestra ningún efecto hasta que volvamos a llamar el SCREEN correspondiente), respectivamente. Naturalmente también puede leer la actual anchura de la pantalla de SCREEN 0 o SCREEN 1 utilizando las instrucciones

```
PRINT PEEK(&HF3AE) o bien
PRINT PEEK(&HF3AF)
```

Otras advertencias:

Al principio ya mencioné, que las modificaciones de la anchura de la pantalla sólo pueden efectuarse dentro de unos límites establecidos (SCREEN 0 = 1 a 40; SCREEN 1 = 1 a 32). Si no se tiene en cuenta esta norma, se muestra el mensaje de error 'Illegal function call'.

Con ayuda de los comandos BASE y VDP puede llamar varias pantallas una tras otra de la VRAM y hojearlas.

vea programa 2 del apéndice

Sin embargo, si aquí sólo pretende indicar pantallas, la WIDTH no le sirve. Tendrá que dirigirse a dichas pantallas directamente con VPOKE.

Hasta producirse un cambio de WIDTH, su ordenador memoriza los últimos valores indicados para SCREEN 0 o SCREEN 1 en lo que respecta a la anchura de la pantalla, incluso si en medio usted cambia a otro SCREEN, por ejemplo la pantalla gráfica SCREEN 2, y después del cambio también actúa en consecuencia.

Para concluir decimos que SCREEN 2 dispone de una constante (anchura de pantalla no modificable) de 32 caracteres (a menos que usted disponga las letras pixel por pixel de forma diferente una al lado de otra, lo que sería posible naturalmente mediante la representación gráfica), mientras que lo mismo no merece la pena en SCREEN 3, puesto que aquí se visualizan tan sólo 8 caracteres por línea (de uso apropiado para títulos).

```
WIDTH 1
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Definicion de un cuadrado"
40 FOR N=1 TO 8
50 READ A$
60 VPOKE 2048+254*8+N-1,VAL("&B"+A$)
70 NEXT N
80 LOCATE 0,10
90 PRINT "y este es su aspecto en SCREEN
 0"
100 PRINT
110 PRINT CHR$(254)
120 FOR N=1 TO 1000
130 NEXT N
140 SCREEN 1
150 PRINT "Definicion de un cuadrado"
160 RESTORE
170 FOR N=1 TO 8
180 READ A$
190 VPOKE 254*8+N-1,VAL("&B"+A$)
200 NEXT N
210 LOCATE 0,10
220 PRINT "y este en SCREEN 1"
230 PRINT
240 PRINT CHR$(254)
250 GOTO 250
260 DATA 11111111
270 DATA 10000001
280 DATA 10000001
290 DATA 10000001
300 DATA 10000001
310 DATA 10000001
320 DATA 10000001
330 DATA 11111111
```

Comandos Gráficos COLOR

Modifica los colores de primer plano, fondo y área de los bordes

vea también los programas del apéndice: COLOR 1
COLOR 2
COLOR 3
COLOR 4
CIRCLE 4

Todos los ordenadores MSX disponen de 15 colores que pueden escogerse utilizando el comando COLOR y cifras entre 0 y 15 correspondientes a primer plano, fondo y área de los bordes. Los colores disponibles son:

0: transparente (se visualiza el color del área de los bordes; aunque este 0 = transparente, en teoría también pueden visualizarse zonas más profundas de la RAM de vídeo
1: negro
2: verde
3: verde claro
4: azul oscuro
5: azul claro
6: rojo oscuro
7: azul celeste
8: rojo
9: rojo claro
10: amarillo oscuro
11: amarillo claro
12: verde oscuro
13: magenta
14: gris
15: blanco

Puesto que a cada color le corresponde un matiz más claro y/o más oscuro, es relativamente fácil representar cuerpos tridimensionales en la pantalla, tal como se muestra en el siguiente ejemplo con un dado:

vea listado de COLOR 1 del apéndice

¡Pruebe usted mismo! Sin embargo, tenga especial cuidado si quiere visualizar más de dos colores al mismo tiempo en un campo de 8*1; en caso contrario pueden producirse fácilmente representaciones poco atractivas de colores mezclados.

vea listado de CIRCLE 4 del apéndice

El comando COLOR se compone de tres parámetros:

COLOR A,B,C

A=color de primer plano (color de escritura)

B=color de fondo (el papel de la pantalla, sobre el cual escribimos con el color de primer plano

C=color de los bordes (mientras no estamos trabajando en SCREEN 0, este parámetro provoca un borde superior e inferior en la pantalla, si no coincide con el color de fondo)

Al conectar un ordenador MSX trabajamos con la combinación de colores prefijada en 15,4,4, es decir, escribimos con letras de color 15 sobre un fondo de color 4, disponiendo además de un borde de color azul oscuro=4 (invisible en SCREEN 0). La pulsación de la tecla F6 (tecla de función con SHIFT) permite volver en cualquier momento a esta combinación inicial de colores (ventaja: blanco sobre fondo azul oscuro es de fácil lectura, es decir, esta combinación ofrece un buen contraste tanto en televisores de color como en los de blanco y negro).

Si deseamos modificar el color de los bordes (tercer parámetro), manteniendo sin embargo los colores de primer plano y de fondo, podemos saltarnos cada uno de estos dos parámetros indicando una coma. Si deseamos un borde de pantalla de color blanco, entramos:

```
10 SCREEN 1
20 COLOR ,,15
```

Antes mencionamos el significado de la tecla de función 6 (devolución al estado inicial de color). Esta necesidad se manifiesta cuando escogemos el mismo color para el primer plano y para el fondo:

```
COLOR 15,15
```

Puede hacer lo que quiera; no puede verse nada. Pulse ahora la tecla F6 y todo vuelve a su normalidad (incluso verá las entradas que haya tecleado antes) puesto que COLOR 15,15 no significa el borrado de la pantalla, sino sólo que tanto el primer plano como el fondo han tomado el mismo color (=blanco). De esta forma, con igual color de fondo y de primer plano puede confeccionar una imagen cualquiera de texto a través de un programa, y cambiar a los colores adecuados de primer plano y de fondo cuando la haya terminado. El efecto es el siguiente: La pantalla ya no se va formando despacio línea por línea ante los ojos del espectador, sino que se crea instantáneamente.

vea listado de COLOR 2 del apéndice

Veamos ahora la diferenciación de los comandos COLOR al aplicarlos en los diversos SCREENs y sus efectos resultantes:

SCREEN 0: permite cambiar sólo los colores de primer plano y de fondo. La función COLOR es ejecutada inmediatamente y no precisa de ningún comando adicional (por ejemplo CLS). Sin embargo, no es posible tampoco producir más de dos colores simultáneos en SCREEN 0 (lo cual sería imposible, puesto que con el comando BASE ya comentamos que SCREEN 0 no dispone de lista de colores en la RAM de vídeo). Al cambiar los colores de primer plano y de fondo, no sólo cambia el color de los caracteres que se entrarán a continuación, sino también el de los ya existentes en la pantalla.

SCREEN 1: Puesto que este SCREEN ya dispone de borde de pantalla, en este caso podemos utilizar los tres parámetros del comando COLOR (colores de primer plano, fondo y área de los bordes). Todos ellos provocan un cambio instantáneo del color (no es necesario entrar antes el comando CLS del BASIC MSX). Si indicamos color de fondo 0, se transparenta el borde de la pantalla, es decir, el área de los bordes se convierte en fondo de la pantalla. El comando COLOR A,B,C no tiene ningún efecto sobre los grupos de ocho caracteres que hayamos podido señalar anteriormente de diversos colores en la RAM de vídeo (puesto que esta información se encuentra en la tabla de colores de la VRAM); únicamente afecta a entradas posteriores de caracteres no definidos en cuanto al color.

SCREEN 2 y 3: Durante la ejecución de un programa tan sólo puede cambiarse el color de las áreas del borde (tercer parámetro). Sin embargo, la modificación del color de primer plano en las pantallas gráficas 2 y 3 se hará visible cuando entremos una nueva representación gráfica o una nueva imagen de texto en nuestro ordenador MSX. Los caracteres entrados anteriormente no se verán afectados por el cambio del primer parámetro del comando COLOR, a diferencia de la reacción inmediata en las pantallas de texto.

Por el contrario, todos los comandos excepto PRESET toman inmediatamente el actual color del primer plano (PRESET toma el color actual del fondo de la pantalla), a menos que asignemos un color diferente al propio comando gráfico, por ejemplo:

vea listado de COLOR 3 del apéndice

Mientras que en la línea 30 el color escogido por COLOR también sirve para el gráfico (color 1), la línea 40 (circunferencia mayor) utiliza el color (15=blanco) escogido especialmente.

Si se cambia el color directamente en el comando gráfico, sólo será válido para este gráfico específico (a

continuación se volverá a utilizar el valor de COLOR entrado anteriormente; vea la línea 50). Si entramos, en cambio, un nuevo color mediante el comando COLOR, este último se mantendrá invariable hasta nueva indicación.

Si se modifica el parámetro número 2 (color de fondo) mediante el comando COLOR, el cambio de color no se efectuará hasta borrar la pantalla seguidamente con ayuda de CLS o SCREEN (para diferenciar los efectos de ambos comandos MSX, consulte en CLS o SCREEN). Después del borrado, el nuevo color de fondo también se mostrará en las pantallas gráficas.

Ahora un par de consejos y trucos importantes:

1) Importante para los programadores en lenguaje máquina: si no desea o no puede utilizar el comando COLOR, modifique los colores de primer plano, fondo y área de los bordes POKEando en las siguientes posiciones de memoria:

primer plano: &HF3E9
fondo: &HF3EA
bordes: &HF3EB

También es posible cambiar los colores de primer plano y fondo (en SCREEN 0) o bien los colores de primer plano y área de los bordes (en SCREEN 1 a 3) a través de una entrada directa en el registro 7 del VDP: para conseguir un primer plano blanco y fondo negro (en SCREEN 0) o bien un primer plano blanco y los bordes de color negro (en SCREEN 1 a 3), teclee simplemente:

```
VDP(7)=&H1F
```

2) En las pantallas gráficas, la tecla CLS/HM o el comando CHR$(12) no tienen el mismo significado que el comando CLS del BASIC MSX para limpiar la pantalla (y con ello para el posible cambio del color de fondo en las pantallas gráficas).

vea listado de COLOR 4 del apéndie

Es decir, la tecla CLS/HOME y el comando PRINT CHR$(12) realizan su función del borrado de pantalla únicamente en las pantallas de texto SCREEN 0 y SCREEN 1.

COLOR 1

```
10 COLOR 15,1
20 SCREEN 2
30 IF Z=1 THEN C=1 ELSE C=3
40 DRAW "bm 112,96 c=c; r32 u32 l32 d32"
50 PAINT(140,86),3
60 IF Z=1 THEN C=1 ELSE C=12
70 DRAW "bm 144,96 c=c; e32 u32 n g32 l3
2 g32 r32 d32"
80 PAINT(154,76),12
90 PAINT(120,60),12
100 IF Z=1 THEN C=1 ELSE C=2
110 DRAW "bm112,96 c=c; r32 g16 l32 e16"
120 PAINT(116,103),2
130 IF Z=1 THEN GOTO 150 ELSE Z=1
140 GOTO 30
150 GOTO 150
```

COLOR 2

```
10 KEY OFF
20 COLOR 15,15
30 SCREEN 0
40 WIDTH 40
50 FOR N=1 TO 220
60 PRINT "MSX ";
70 NEXT N
80 COLOR 1,15
```

COLOR 3

```
10 COLOR 1,4
20 SCREEN 2
30 CIRCLE(128,96),20
40 CIRCLE(128,96),40,15
50 CIRCLE(128,96),60
60 GOTO 60
```

COLOR 4

```
10 COLOR 1,4
20 SCREEN 2
30 CIRCLE(128,96),20
40 PRINT CHR$(12)
50 GOTO 50
```

Comandos Gráficos TAB

Tabula la emisión de caracteres

vea también los programas del apéndice: TAB 1
TAB 2
TAB 3

Seguramente reconocerá la función tabuladora tal como se encuentra en cualquier máquina de escribir: pulsar una tecla que mueva el carro de impresión un determinado número de caracteres hacia la derecha. Si además la distancia tabulada es programable, podrá confeccionar una tabla regular con su máquina de escribir.

Su ordenador MSX también le ofrece un control parecido de tabulación: pulse la tecla que contenga las letras TAB o una doble flecha hacia la derecha, y el cursor se moverá en la pantalla ocho caracteres hacia la derecha.

Sin embargo, su ordenador MSX dispone en su BASIC además de un tabulador de definición libre, que podrá ser llamado dentro de una instrucción PRINT directamente mediante la secuencia de letras TAB. En contraposición a la anterior pulsación única de una tecla, el comando BASIC TAB precisará además de un valor numérico:

```
PRINT TAB(10);"MSX"
PRINT TAB(20);"MSX"
```

La cifra indicada detrás del término TAB indica la distancia a partir de la última posición del cursor. Para ello, su ordenador MSX siempre parte de la siguiente posición de emisión a realizar sin TAB y le suma correspondientemente los caracteres indicados. Sin embargo, si la impresión no comienza en el extremo izquierdo de la línea, la emisión TAB se efectúa siempre calculando a partir de la actual posición del extremo izquierdo de la línea (extremo izquierdo de línea = TAB(0)). Vea los tres ejemplos siguientes:

```
10 CLS
```

```
20 PRINT TAB(10);"MSX"
```

La emisión comienza en la posición 10+1 del cursor en la línea 1

```
10 CLS
20 LOCATE 10,0
30 PRINT TAB(15);"MSX"
```

La emisión comienza en la posición 15+1 del cursor en la línea 1

```
10 CLS
20 LOCATE 10,0
30 PRINT TAB(5);"MSX"
```

La emisión comienza en la posición 10+1 del cursor en la línea 1, puesto que TAB(5) es ignorado por su reducido valor comparado con el LOCATE 10,0 precedente.

Advertencia: en las explicaciones sobre estos tres programas cortos hemos sumado '1' a la posición del cursor, puesto que el ordenador MSX cuenta incluyendo el 0.

El valor después de la función TAB puede estar comprendido entre 0 y 255, avanzando el cursor varias líneas sin ningún problema. La entrada

```
PRINT TAB(0);"MSX"
```

significaría que el comando TAB efectúa una función adicional a la instrucción PRINT, es decir, la emisión de caracteres comienza por el margen izquierdo.

Tras graduar previamente la anchura de la pantalla a 40 caracteres (WIDTH 40), la instrucción

```
PRINT TAB(40);"MSX"
```

ordena que la emisión comience dos líneas más abajo

próxima línea: TAB(0) a TAB(39)
dos líneas más adelante: TAB(40) a TAB(79)

Sin embargo, no debe olvidarse que el comando TAB admite tan sólo números entre 0 y 255. Un número mayor de 255 provocaría el mensaje de error 'Illegal function call'.

Parece que con ayuda del comando TAB sea muy fácil confeccionar una emisión formateada. Veamos para ello el ejemplo siguiente:

<u>vea listado de TAB 1 del apéndice</u>

A primera vista parece que la salida a pantalla esté formateada, pero al observar más detenidamente los valores numéricos alineados verticalmente veremos, que las unidades de un número se encuentran directamente debajo de las decenas y centenas de otros. De esta forma no puede verse claramente la mayor o menor dimensión de dichos números, ni tampoco sirve para usos matemáticos.

Podemos ampliar nuestro programa indicando, además del comando TAB, la dimensión del número a emitir, o sea su formato. Para ello disponemos del comando PRINT USING: los números emitidos tienen valores entre 1 y 1000, con lo que la representación máxima no superará las cuatro cifras; modificamos la línea correspondiente de nuestro programa, ampliándola con PRINT USING ''≠≠≠''.

<u>vea listado de TAB 2 del apéndice</u>

Ahora la representación es limpia y clara; los comandos TAB y PRINT USING cooperan estrechamente para cumplir su función.

Otras advertencias para el uso correcto del comando TAB:

1) No sólo puede utilizar TAB para la pantalla en combinación con PRINT, sino también para la impresora junto con el comando LPRINT correspondiente.

2) Para continuar escribiendo en la misma línea con PRINT TAB tras una instrucción INPUT, hay que desplazar el cursor una línea hacia arriba, utilizando la instrucción:

LOCATE distancia deseada, CSRLIN-1

tal como le muestra el ejemplo siguiente:

vea listado de TAB 3 del apéndice

Ejecutando la instrucción TAB de forma que deba saltarse caracteres ya escritos, éstos se borrarán en SCREEN 0 y 1, mientras que se mantienen en las pantallas gráficas SCREEN 2 y SCREEN 3.

TAB 1

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 20
50 PRINT INT(RND(1)*2000);
60 PRINT TAB(10);INT(RND(1)*2000);
70 PRINT TAB(20);INT(RND(1)*2000);
80 PRINT TAB(30);INT(RND(1)*2000)
90 NEXT N
```

TAB 2

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 20
50 PRINT USING "####";INT(RND(1)*2000);
60 PRINT TAB(10);USING "####";INT(RND(1)
*2000);
70 PRINT TAB(20);USING "####";INT(RND(1)
*2000);
80 PRINT TAB(30);USING "####";INT(RND(1)
*2000)
90 NEXT N
```

TAB 3

```
10 SCREEN 0
20 WIDTH 40
30 COLOR 1,15
40 PRINT "Entre solo numeros inferiores
a 100"
50 PRINT
60 INPUT "Entre un numero ";A
70 IF A>99 THEN GOTO 60
80 LOCATE 25,CSRLIN-1
90 PRINT "cuadrado:";A*A
100 GOTO 60
```

Comandos Gráficos LOCATE

Mueve el cursor hacia coordenadas determinadas en pantallas de texto

vea también los programas del apéndice: LOCATE 1
LOCATE 2
LOCATE 3
LOCATE 4
LOCATE 5
LOCATE 6
LOCATE 7
LOCATE 8
LOCATE 9
LOCATE 10
LOCATE 11

Con ayuda del comando LOCATE del MSX podemos conseguir dos objetivos diferentes:

1) Dirigir el cursor hacia una posición determinada de las pantallas de texto (SCREEN 0 y SCREEN 1)

2) Activar y desactivar el cursor de texto

Con respecto a 1): para obtener una configuración agradable de la pantalla, no siempre es conveniente comenzar la emisión del texto en el extremo izquierdo de la pantalla, tal como se muestra en el siguiente ejemplo:

vea listado de LOCATE 1 del apéndice

Un primer paso para mejorar la visualización del texto consiste en introducir espacios y líneas vacías:

vea listado de LOCATE 2 del apéndice

Sin embargo, esta solución no es la ideal, porque no sólo gasta mucha memoria del ordenador inútilmente, sino que además reduce la claridad del listado del programa. Si bien es verdad que el comando TAB nos ayuda a repartir mejor el

nivel horizontal:

vea listado de LOCATE 3 del apéndice

para dejar líneas libres en sentido vertical, sin embargo, deben utilizarse líneas PRINT vacías. Tampoco resulta muy efectivo, y menos aún cuando pretendemos saltarnos varias líneas en la pantalla.

Podemos conseguir resultados mucho más satisfactorios con ayuda del comando LOCATE: indicamos simplemente la posición deseada de la pantalla (de forma parecida a la realizada con el comando DRAW "B M 128,96" en la pantalla gráfica para dirigirnos al centro de la pantalla) entrando los valores deseados de X (columna) e Y (línea). Sin embargo, debemos ser conscientes que la cuenta comienza por la posición 0,0, que corresponde al primer carácter en la esquina superior izquierda de la pantalla.

Según la anterior graduación de WIDTH (de 1 a 40 en SCREEN 0, pero sólo de 1 a 32 en SCREEN 1), los valores de la posición X (columna) pueden oscilar entre 1 y 39, y los de la posición Y (línea) entre 0 y 31. Observémoslo en un ejemplo:

vea listado de LOCATE 4 del apéndice

O de forma más clara:

vea listado de LOCATE 5 del apéndice

A diferencia del comando PRINT sin LOCATE (entrada de espacios, líneas vacías o comandos TAB), que borran la pantalla hasta la posición indicada (PRINT:PRINT:PRINT), el comando LOCATE no tiene ningún efecto sobre líneas y columnas ya existentes de la pantalla, como nos muestra claramente el próximo programa (cada comando PRINT debe ir seguido de un punto y coma para evitar que el ordenador MSX efectúe la emisión con salto de línea incluido, lo que significa tanto como: borra la línea a partir de la emisión

hasta su final):

vea listado de LOCATE 6 del apéndice

Según un presentimiento -todavía se mantiene el listón de las teclas de función- en este programa he limitado el margen de visualización a las líneas 0 a 21. Si hubiese permitido 22 líneas, podría haberse producido algunas veces el scroll de la pantalla (por ejemplo, al moverse casualmente hacia la posición X=38, Y=21, es decir, la emisión hubiera tenido que continuar en la línea 22,lo que significa que se habría producido el scroll ascendente por una línea).

Sin embargo, su ordenador MSX ya es lo suficientemente listo para continuar una emisión comenzada en la línea siguiente ... aunque para ello deba recurrir al scroll de la pantalla.

Usted está libre de sustituir X e Y por valores hasta 255 en el comando LOCATE. Aunque el ordenador MSX también entienda las posiciones ilegales (por ejemplo, X mayor que 39) de la pantalla de texto, en este caso sólo ejecuta el máximo valor representable (por ejemplo, si X=100, la visualización es emplazada en la columna X=39 en SCREEN 0 y WIDTH 40).

Tal como ya hemos mencionado, el comando LOCATE sólo puede utilizarse razonablemente en las dos pantallas de texto (SCREEN 0 y SCREEN 1). Si entra un comando LOCATE mientras se encuentra en una página gráfica, éste no será ejecutado. Tal como el comando LOCATE para las pantallas de texto, para las pantallas gráficas debe utilizar el comando DRAW "B M xl,yl".

Con respecto a 2): A veces es bastante molesto ver continuamente en la pantalla el cursor del texto en un programa que se está ejecutando. Saltando mediante coma la emisión X e Y del comando LOCATE, se borra el cursor:

```
LOCATE ,,0
```

Este efecto puede invertirse, indicando un 1 ó cualquier valor menor que 256 en lugar del 0:

```
LOCATE ,,1
```

El borrado del cursor tiene la siguiente razón: de esta forma puede emitirse información incluso en la última línea de la pantalla, sin que se efectúe automáticamente (y por ello involuntariamente) el scroll ascendente de la misma. Sin embargo, vaya con precaución: sólo puede invalidar el cursor dentro de un programa que se está ejecutando. En el mismo momento de realizar una entrada (INPUT, INPUT$ ...), el cursor aparece de nuevo en la pantalla, tanto si el comando LOCATE precedente lo ha invalidado como si no.

Para demostrarlo juntamos simplemente ambas funciones LOCATE entre sí: la elección de la posición y el borrado del cursor de la pantalla de texto:

vea listado de LOCATE 7 del apéndice

o lo mismo con cursor de posición visible:

vea listado de LOCATE 8 del apéndice

Además también podemos crear un cursor intermitente, al igual que en otros ordenadores, en relación con la subrutina en lenguaje máquina (ON INTERVAL):

vea listado de LOCATE 9 del apéndice

Por cierto:

1) El cursor de la pantalla de texto corresponde a CHR$(255) del juego de caracteres del ordenador, lo que puede demostrarse claramente a través del siguiente programa:

<u>vea listado de LOCATE 10 del apéndice</u>

A primera vista se aprecia una pantalla oscura. Ahora mueva el cursor encima de cada una de las letras del último texto escrito. Inmediatamente se modifica toda la superficie de la pantalla según el actual aspecto del cursor (representación inversa de los caracteres que se encuentran debajo del mismo).

2) Lo que podemos localizar mediante el comando LOCATE (coordenadas X e Y del cursor de texto), también podemos POKEarlo directamente a la memoria:

posición de memoria &HF3DC = posición Y
posición de memoria &HF3DD = posición X

Es decir, la entrada LOCATE siguiente

```
LOCATE 15,10
```

equivale a POKEar en las dos posiciones de memoria:

```
POKE &HF3DD,15
POKE &HF3DC,10
```

En general puede recordar lo siguiente: cualquier comando BASIC puede ser sustituido de una manera u otra por un comando POKE o PEEK. Sin embargo, deberíamos ser sinceros y reconocer: ¡el manejo con instrucciones BASIC es mucho más rápido y sencillo que recordar miles de posiciones de memoria!

A través de las posiciones de memoria (las llamadas direcciones de las variables del sistema) también podemos activar y desactivar el cursor:

cursor desactivado: POKE &HFCA9,0
cursor activado: POKE &HFCA9,1

3) Tal como ya mencioné en otro lugar, podemos dirigirnos directamente a cualquier posicion de la pantalla con ayuda del comando LOCATE x,y para escribir algo en ella. También existe la posibilidad de numerar toda la pantalla desde la posición 0 hasta la 959 (con WIDTH 40). En este caso debemos trabajar directamente en la memoria de vídeo, lo que se demuestra con el siguiente programa:

vea listado de LOCATE 11 del apéndice

En el centro de la pantalla aparece ahora una vertical que conduce línea por línea (STEP 40) hasta el extremo inferior de la pantalla.

LOCATE 1

```
10 SCREEN 0
20 COLOR 1,15
30 PRINT "Menu"
40 PRINT "1) Entrar direcciones"
50 PRINT "2) Buscar direcciones"
60 PRINT "3) Almacenar direcciones"
```

LOCATE 2

```
10 SCREEN 0
20 COLOR 1,15
30 PRINT "          Menu
31 PRINT
40 PRINT "          1) Entrar direcciones"
50 PRINT "          2) Buscar direcciones"
60 PRINT "          3) Amacenar direcciones"
```

LOCATE 3

```
10 SCREEN 0
20 COLOR 1,15
30 PRINT TAB(15) "Menu"
31 PRINT
40 PRINT TAB(10) "1) Entrar direcciones"
50 PRINT TAB(10) "2) Buscar direcciones"
60 PRINT TAB(10) "3) Almacenar direccion
es"
```

LOCATE 4

```
10 SCREEN 0
20 COLOR 1,15
30 LOCATE 15,5
40 PRINT "Menu"
50 LOCATE 10,7
60 PRINT "1) Entrar direcciones"
70 LOCATE 10,8
80 PRINT "2) Buscar direcciones"
90 LOCATE 10,9
100 PRINT "3) Almacenar direcciones"
```

LOCATE 5

```
10 SCREEN 0
20 COLOR 1,15
30 LOCATE 15,5
40 PRINT "Menu"
50 FOR N=7 TO 9
60 LOCATE 10,N
70 READ A$
80 PRINT A$
90 NEXT N
100 DATA 1) Entrar direcciones,2) Buscar
 direcciones,3) Almacenar direcciones
```

LOCATE 6

```
10 SCREEN 0
20 COLOR 1,15
30 FOR N=1 TO 100
40 LOCATE INT(RND(1)*39),INT(RND(1)*21)
50 PRINT "MSX";
60 NEXT N
```

LOCATE 7

```
10 SCREEN 0
20 COLOR 1,15
30 LOCATE 0,0,0
40 PRINT "MSX"
50 GOTO 50
```

LOCATE 8

```
10 SCREEN 0
20 COLOR 1,15
30 LOCATE 0,0,1
40 PRINT "MSX"
50 GOTO 50
```

LOCATE 9

```
10 SCREEN 0
20 COLOR 1,15
30 ON INTERVAL=10 GOSUB 60
40 INTERVAL ON
50 GOTO 50
60 IF Z=1 THEN Z=0 ELSE Z=1
70 ON Z+1 GOTO 80,100
80 LOCATE ,,1
90 RETURN
100 LOCATE ,,0
110 RETURN
```

LOCATE 10

```
10 WIDTH 40
20 SCREEN 0
30 COLOR 1,15
40 FOR N=1 TO 960
50 PRINT CHR$(255);
60 NEXT N
70 PRINT "MSX es SOBERBIO"
```

LOCATE 11

```
10 WIDTH 40
20 SCREEN 0
30 COLOR 1,15
40 FOR N=20 TO 940 STEP 40
50 VPOKE N,33
60 NEXT N
```

Comandos Gráficos POS

Lee la posición columna en las pantallas de texto

vea también los programas del apéndice: POS 1
POS 2
POS 3
POS 4
POS 5
POS 6

Existen varias posibilidades de determinar la posición del cursor en la pantalla de texto, o bien de leer su posición actual. Por un lado, con el comando LOCATE (X,Y) podemos indicar las coordenads X e Y, dónde debe efectuarse la próxima entrada PRINT o INPUT. El comando CSRLIN (en realidad es una variable de sistema cuyo valor es leído sencillamente) sirve para determinar además la línea de la pantalla de texto donde se encuentra actualmente el cursor.

Es decir, mientras CRSLIN lee la coordenada Y del cursor, el comando POS (o mejor dicho, la variable de sistema POS) sirve para leer la correspondiente posición del carácter en dicha línea, informando sobre la coordenada X del cursor.

Veamos la forma de efectuarse la emisión de la variable de sistema POS:

vea listado de POS 1 del apéndice

Al ejecutar el programa con RUN, la pantalla se llena con el término "MSX" en cuestión de pocos segundos. Detrás de cada "MSX" se muestra un número que indica la actual posición del carácter. Si queremos conocer también la línea en que nos encontramos en ese momento, leemos además su correspondiente número con CSRLIN:

```
45 PRINT CSRLIN;
```

¿Qué uso razonable pueden tener las emisiones de las coordenadas X e Y mediante POS y CSRLIN en la pantalla de

texto? Intentaré explicárselo con la ayuda de dos ejemplos prácticos:

vea listado de POS 2 del apéndice

Al utilizar el comando POS hay que fijar la cantidad de caracteres por línea que pueden ser visualizados y el SCREEN en que se está trabajando (WIDTH 40 y SCREEN 0). El texto en la línea 50 del ejemplo llega hasta la columna 27. Si la entrada de su nombre excede del espacio disponible de caracteres, se avanza una línea mediante PRINT para continuar después con la emisión de su nombre en la siguiente línea. Esto significa, que puede emplear el comando POS(...) para obtener una visualización limpia en la pantalla cuando las entradas son variables (puesto que existen, como en nuestro ejemplo, nombres tan largos como 'José María González' y también otros más cortos como 'Luis Pi'; nuestro pequeño programa produciría la visualización adecuada con ayuda del comando POS).

Otra aplicación de POS hace referencia a la emisión formateada en la pantalla:

vea listado de POS 3 del apéndice

En este caso, el término "MSX" sólo se muestra al llegar a determinada columna de la pantalla. Algo parecido podría emplearse para la emisión formateada de una hoja de cálculo. Sin embargo, debería añadir otro comando: PRINT USING le permite emitir números de dimensión prefijada alineados por la derecha, mientras que el comando POS sólo provoca una alineación por la derecha de las primeras cifras de todos los números a emitir, lo que no corresponde a una alineación matemática:

vea listado de POS 4 del apéndice

y actuando junto con PRINT USING, el resultado (más limpio) es el siguiente:

<u>vea listado de POS 5 del apéndice</u>

Mientras que con CSRLIN es suficiente mostrar esta variable simplemente en la pantalla con PRINT, el comando, o mejor dicho, la variable POS requiere un argumento adicional. El número indicado puede ser cualquiera, siendo el resultado siempre idéntico; el argumento permite darle una configuración del tipo POS(N).

Antes he mencionado que, en lugar de utilizar comandos que corresponden a los valores de variables de sistema (como POS(N) y CSRLIN), también podemos leer directamente en la memoria:

valor actual de POS(N) en posición $HF3DD
valor actual de CSRLIN en posición &HF3DC

Tenga en cuenta, que POS(N) y CSRLIN comienzan a contar a partir de 0, mientras que la indicación de las dos posiciones anteriores de la memoria RAM comienza con 1.

¿Existe también un comando en las pantallas gráficas, que permita leer la posición actual del cursor? No, no existe ningún comando específico para ello. En su lugar pueden leerse cuatro valores diferentes en la memoria:

valor absoluto de la coordenada X en posición &HFCB3
valor absoluto de la corrdenada Y en posición &HFCB5
valor relativo de la coordenada X en posición &HFCB7
valor relativo de la coordenada Y en posición &HFCB9

<u>vea listado de POS 6 del apéndice</u>

Dos ejemplos numéricos servirán para ilustrarlo:

1) Línea 30: DRAW "N R8"

&HFCB3: 136 &HFCB5: 96 &HFCB7: 128 &HFCB9: 96

2) Línea 30: CIRCLE(128,96),20

&HFCB3: 20 &HFCB5: 96 &HFCB7: 128 &HFCB9: 96

La diferencia entre absoluto y relativo se debe a que muchos comandos gráficos parten de un punto determinado de la pantalla, mientras que el dibujo es representado en otro relativo al punto de partida (por ejemplo, indicación del centro de la circunferencia, mientras que la emisión se efectúa por el radio; una vez terminada la circunferencia, el cursor gráfico vuleve al punto absoluto (centro de la misma)).

En la pantalla gráfica, el comando LOCATE X-Y de la pantalla de texto es sustituido por DRAW "B M X,Y" o por PRESET(X,Y).

Otras dos advertencias: muchos comandos gráficos permiten dirigirse también a coordenadas fuera de la ventana de la pantalla. Al leer el comando POS(N) se muetra el valor -1.

Aunque no pueda utilizarse directamente el comando POS(N) en las pantallas gráficas, incluso ahí informa sobre la posición columna del cursor en la última pantalla de texto (sea SCREEN 0 o SCREEN 1).

POS 1

```
10 SCREEN 0
20 COLOR 1,15
30 PRINT "MSX";
40 PRINT POS(1);
50 GOTO 30
```

POS 2

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 INPUT "Escriba su nombre, por favor
         ";A$
50 PRINT "Siento especial satisfaccion p
or poder  saludar a: Senor/Senora ";
60 IF LEN(A$)>40-POS(0) THEN PRINT:PRINT
 A$ ELSE PRINT A$
```

POS 3

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 1000
50 IF POS(1)/4 =INT(POS(1)/4) THEN PRINT
 "MSX ";
60 NEXT N
```

POS 4

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 1000
50 IF POS(1)/5 =INT(POS(1)/5) THEN PRINT
 INT(RND(1)*999);ELSE PRINT " ";
60 NEXT N
```

POS 5

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 1000
50 IF POS(1)/5 =INT(POS(1)/5) THEN PRINT
 USING "###";INT(RND(1)*999);ELSE PRINT
" ";
60 NEXT N
```

POS 6

```
10 SCREEN 2
20 PSET(128,96)
30 DRAW"N R8
40 A=PEEK(&HFCB3)
50 B=PEEK(&HFCB5)
60 C=PEEK(&HFCB7)
70 D=PEEK(&HFCB9)
80 SCREENO
90 PRINT A;B;C;D
100 PRINT
110 LIST
```

Comandos Gráficos LPOS

Lee la posición actual de la cabeza impresora

vea también el programa del apéndice: LPOS 1

Este comando (o sea, la lectura de esta variable) informa sobre la posición (columna) donde se encuentra actualmente la cabeza impresora de la impresora conectada.

Esto es importante especialmente para emitir tablas por la impresora, deseando que todos los datos se muestren alineados verticalmente (como es normal al imprimir una tabla).

El siguiente programa ejecuta esta función al mismo tiempo en la pantalla y en la impresora:

vea listado de LPOS 1 del apéndice

Tanto en la pantalla como en la impresora (LPOS(1)) se emiten tres veces dos números alineados verticalmente en forma de tabla.

Otra aplicación del comando LPOS(1) consiste en consultar a la impresora, teniendo un texto emitido, si en la línea escogida queda suficiente espacio para colocar la siguiente palabra; de lo contrario debe efectuarse un avance de línea con ayuda del comando LPRINT.

Por cierto:

1) También podemos leer en LPOS(1) (que significa Line Print X POSition), efectuando en la memoria de variables de sistema del MSX directamente la función siguiente:

```
PRINT &HF415
```

Al trabajar directamente con la posición de memoria &HF415, también podemos asignar un valor al parámetro LPOS(1), ejecutando de esta forma un tipo de comando LOCATE:

```
POKE &HF415,40
```

o sea, la cabeza impresora se encuentra ahora en la posición 40.

2) Otra forma mejor y más razonable que el comando LPOS(1) (o sea, la variable de la memoria) para dirigir la cabeza impresora, suele consistir casi siempre en dirigir la salida a través de LPRINT TAB, el comando LPRINT USING o bien utilizando la instrucción combinada LPRINT USING TAB (vea ejemplos en TAB).

Otra cosa sucede con el control de la cabeza impresora para impedir que la impresión continúe en la siguiente línea a partir del final de la anterior.

LPOS 1

```
10 FOR N=1 TO 3
20 READ A,B
30 IF POS(1)<>5 THEN PRINT " ";:GOTO 30
40 PRINT A;
50 IF LPOS(1)<>5 THEN LPRINT " ";:GOTO 5
0
60 LPRINT A;
70 IF POS(1)<>20 THEN PRINT " ";:GOTO 70
80 PRINT B;
90 IF LPOS(1)<>20 THEN LPRINT " ";:GOTO
90
100 LPRINT B;
110 PRINT
120 LPRINT
130 NEXT N
140 DATA 10,20,30,40,50,60
```

Comandos Gráficos CSRLIN

Lee la posición línea del cursor en las pantallas de texto

vea también el programa del apéndice: CSRLIN 1

Este comando le permite conocer la línea de la pantalla donde se encuentra actualmente el cursor de texto (sólo aplicable en SCREEN 0 y SCREEN 1).

Ello permite evitar que se reescriba involuntariamente en una zona de la pantalla confeccionada anteriormente, lo que se demuestra con un ejemplo: en la línea 10 de la pantalla escribimos el término 'MSX'. A continuación ordenamos a nuestro MSX reescribir en toda la pantalla la palabra 'SUPER', excepto en la línea 10 (aunque en la pantalla pueda reconocerse fácilmente la línea 10 contándolas, el ordenador MSX sin embargo la considera como línea 9, puesto que cuenta 0, 1, 2, ... 9 en lugar de 1, 2, 3, ...10):

vea listado de CSRLIN 1 del apéndice

Es decir, el ordenador MSX comprueba en qué línea de la pantalla se encuentra; cuando está seguro de no escribir encima de algo ya existente (IF CSRLIN 9) efectuará su cometido (THEN PRINT "SUPER").

En las pantallas de texto (SCREEN 0 y SCREEN 1) el comando CSRLIN sólo puede utilizarse dentro del intervalo 0 a 22 (si la indicación de las teclas de función aparece en la línea 23) o de 0 a 23 (si dicha indicación está invalidada). Una vez que el texto haya desaparecido de la pantalla (scroll), éste se habrá perdido irremediablemente.

La aplicación del comando CSRLIN comienza a ser razonable cuando es utilizado, por ejemplo, en un programa propio de entrada de texto. Para ello, sin embargo, además de la coordenada Y (mediante CSRLIN) también habrá que determinar la coordenada X del cursor en la pantalla (mediante POS), así como ser capaz de moverlo hacia cualquier posición de la pantalla (mediante LOCATE).

Por cierto: la coordenada Y del cursor (línea de la pantalla determinada a través de CSRLIN) también puede ser leída en la posicion &HF3DD de la memoria.

Si además queremos conocer la coordenada Y del cursor también en la pantalla gráfica, tendremos que consultar a otra posicion de memoria, la &HFCB5.

Para leer la coordenada Y del cursor gráfico existe además la posición &HFCB9, que informa sobre la distancia entre el cursor y su punto de partida en un direccionamiento relativo (por ejemplo, con ayuda de STEP en los comandos CIRCLE o LINE).

CSRLIN 1

```
10 SCREEN 0
20 COLOR 1,15
30 LOCATE 0,9
40 PRINT "MSX"
50 FOR M=1 TO 1000
60 NEXT M
70 LOCATE 0,0
80 FOR N=0 TO 20
90 IF CSRLIN<>9 THEN PRINT "SUPER" ELSE
LOCATE 0,CSRLIN+1
100 NEXT N
```

Comandos Gráficos CHR$

Direccionamento directo al juego de caracteres (0 a 255)

vea también los programas del apéndice: CHR$ 1
CHR$ 2
CHR$ 3
CHR$ 4
CHR$ 5

Si ha seguido el orden alfabético de lectura hasta ahora, ya conocerá el comando CHR$ de la descripción del comando ASC: mientras que este último convierte cualquier carácter representable en un número decimal (su ordenador MSX trabaja en su interior precisamente con este número, no con el carácter), el comando CHR$ tiene justamente el efecto contrario. Al entrar:

```
PRINT CHR$(65)
```

resultado: A

De esta forma podemos visualizar el juego completo de caracteres (que puedan representarse) del ordenador MSX, a través del siguiente programa:

vea listado de CHR$ 1 del apéndice

Al principio se muestran algunos caracteres de puntuación (punto, coma, ...), después las cifras 0 a 9, a continuación las letras mayúsculas y minúsculas, y finalmente también los caracteres especiales internacionales (ü, ...), para terminar con algunos caracteres matemáticos y gráficos. Se preguntará seguramente, por qué no hemos comenzado con el número 1, o incluso el 0, al visualizar el juego de caracteres en la pantalla (vea CHR$ 1). La razón es la siguiente: los primeros 32 caracteres no pueden visualizarse de forma legible, al menos en lo que respecta a la emisión en pantalla. Ejecute lo siguiente para verlo:

```
PRINT CHR$(7)
```

Tal como puede escuchar, su ordenador (es decir, el altavoz del televisor conectado a su MSX emite sonidos), suena un sonido BEEP (vea también BEEP) y en la pantalla reaparece la combinación de letras harto conocida: OK. Esto significa, que con CHR$(7), por ejemplo, no se efectúa ninguna emisión visible en la pantalla.

Qué tal si probase con:

```
PRINT CHR$(13)
```

Aquí tampoco sucede nada en la pantalla a primera vista (aparentemente). Tan sólo se emite una línea vacía adicionalmente en la pantalla.

Insistimos en estos dos caracteres para mostrar la utilidad de su aplicación en casos aislados: para ello teclee el siguiente programa:

vea listado de CHR$ 2 del apéndice

Al ejecutar este programa con RUN y obedecer las instrucciones, no sucede nada especial. Sin embargo, si entra un número mayor que 100, en contra de lo exigido en la línea 30, su ordenador MSX emite un corto BEEP, avisando de esta forma -además de la frase que aparece en pantalla ('Por favor, siga mis instrucciones')- que la entrada no ha sido correcta (ambas instrucciones en línea 40).

En contrapartida, CHR$(13) simula la pulsación de la tecla ENTER. Hagamos una prueba práctica, modificando la ocupación de las teclas de función 1 y 2 de la siguiente forma:

```
KEY 1,"LIST"
KEY 2,"LIST"+CHR$(13)
```

Si conserva en la memoria todavía el programa anterior, compruebe ahora las dos teclas de función (1 y 2): después de F1 tendrá que pulsar además la tecla ENTER (para ejecutar el comando), mientras que la ejecución de la tecla F2 se

efectúa automáticamente (gracias al CHR$(13) añadido).

Hay otra instrucción que exige utilizar el comando CHR$: la asignación de la variable SPRITE$ determina el aspecto de sprites. Si no somos exigentes y nos conformamos con un patrón de raya única, podemos indicarlo directamente a través del teclado:

```
SPRITE$(0)="UUUUUUUU"
```

Sin embargo, si el patrón del sprite debe contener además dos líneas vacías en su centro (puntos desactivados), debería entrarse CHR$(0) a través del teclado; no es tan fácil. En lugar de la difícil entrada directa a través de teclado, puede utilizarse simplemente el comando CHR$:

```
SPRITE$(0)="UUU"+CHR$(0)+CHR$(0)+"UUU"
```

Es posible entrar secuencias de caracteres muy diversas a través del teclado, pero es más difícil leerlas a partir de un programa. Especialmente en el sistema operativo, que puede activarse en los ordenadores MSX con una unidad de disco conectada, se prefiere utilizar la combinación de las teclas CTRL junto con una letra cualquiera. Sin embargo, ¿de qué forma podemos saber en un programa BASIC de confección propia, si alguien ha pulsado una tecla combinada con CTRL? Para ello se dispone de una fórmula de programa muy sencilla con ayuda del comando BASIC CHR$ tratado en este capítulo:

vea listado de CHR$ 3 del apéndice

En este programa no sucede nada mientras no se pulse ninguna tecla junto con la CTRL. Pero cuando entre, por ejemplo, CTRL-E (borrar hasta el final de la línea), el ordenador muestra en la pantalla que ha entendido la indicación correctamente.

En algunas impresoras, el comando CHR$ sirve para escoger diversos tipos de letra. Al menos en las impresoras EPSON, la salida hacia impresora comienza con el comando ESC, que corresponde al número 27 del carácter CHR$. Si desea, por ejemplo, graduar impresión reforzada (Emphasized mode), entre el siguiente carácter de control de impresora:

```
PRINT CHR$(27);"E"
```

Finalmente, una última aplicación del comando CHR$ (existen infinidad de ellas, que usted mismo podrá desarrollarse después de trabajar intensamente este capítulo: el programa exige una entrada en mayúsculas, pero no siempre es seguro que ello se cumpla realmente (ya sabrá seguramente que el ordenador MSX distingue entre las letras minúsculas y mayúsculas). Convertimos pues sencillamente la entrada hecha en minúsculas en letra mayúscula, si no nos gusta tal como es:

vea listado de CHR$ 4 del apéndice

Después de entrar cualquier palabra, en mayúsculas o minúsculas (o ambas), en cualquier caso a continuación, la palabra se emitirá en mayúsculas en la pantalla.

Otras dos advertencias: como es habitual, hay que tener especial precaución con el número mágico 255. Para evitar un mensaje de error (Illegal function call), el número utilizado en el comando CHR$ debe encontrarse entre 0 y 255.

CHR$(255) corresponde al carácter del cursor. Al indicarlo puede crearse un efecto simpático:

vea listado de CHR$ 5 del apéndice

Puesto que el cursor se muestra repetidas veces en la pantalla, la modificación del carácter del cursor también se efectúa simultáneamente en todos aquellos lugares donde éste aparece; de esta forma se explica el rápido cambio del contenido de la pantalla, aunque en el producto final se relaciona simplemente con el cambio rápido de visualización de un carácter.

CHR$ 1

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 FOR N=32 TO 255
50 PRINT CHR$(N);
60 NEXT N
```

CHR$ 2

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40:INPUT "Por favor, entre un
                        numero inferior a 100
 ";A
40 IF A>99 OR A<0 THEN PRINT "Por favor,
 siga mis instrucciones";CHR$(7)
50 GOTO 30
```

CHR$ 3

```
10 CLS
20 A$=INKEY$
30 IF A$="" THEN GOTO 20
40 IF ASC(A$)<32 THEN GOTO 50 ELSE GOTO
20
50 PRINT " Usted ha pulsado CTRL-";CHR$(
ASC(A$)+64)
60 GOTO 20
```

CHR$ 4

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 Z=96
50 INPUT "Escriba una palabra, por favor
 ";A$
60 FOR N=1 TO LEN(A$)
70 IF MID$(A$,N,1)>="a"THEN MID$(A$,N,1)
=CHR$(ASC(MID$(A$,N,1))-32)
80 NEXT N
90 PRINT A$
```

CHR$ 5

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 23
50 PRINT STRING$(40,255);
60 NEXT N
80 PRINT "Por favor, lleve el cursor enc
ima        de las letras esritas"
```

Comandos Gráficos ASC

Convierte los caracteres en sus correspondientes valores numéricos

vea también los programas del apéndice: ASC 1
ASC 2
ASC 3

Como ya es sabido, el ordenador MSX representa cada carácter (letras, cifras o gráficos) internamente con los números '0' y '1', que corresponde a los estados 'apagado' y 'encendido'.

El comando ASC sirve para representar en la pantalla como números decimales los caracteres entrados a través del teclado:

```
10 PRINT ASC("A")

resultado: 65
```

Puesto que en este caso su ordenador MSX no considera la letra A como variable o cajón, hay que indicarla entre comillas (para señalizarla como string).

Sin embargo, también podemos hacer que el ordenador calcule una variable de cadena directamente. Aunque sólo esté compuesta por un único carácter, el proceso seguido es el mismo que el representado en el ejemplo anterior. Si el string es más largo, el ordenador MSX interpreta tan sólo el primer carácter:

```
10 A$ = "CASA"
20 PRINT ASC(A$)

resultado: 72
```

Si deseamos ver la representación original de la letra H (en forma de 0 y 1) en la pantalla, escogeremos otra forma más compleja de visualización, pero que con ayuda del vacabulario BASIC MSX no es tan difícil de manejar:

```
10 A$ = "CASA"
20 PRINT BIN$(ASC(A$))
```

resultado: 1000100

Vayamos ahora a aplicar el comando ASC en nuestro programa:

1) Deseamos reconocer entradas del teclado que no pueden ser visualizadas directamente en la pantalla.

2) Deseamos emitir caracteres gráficos de forma legible en una impresora que no sea del MSX.

3) Deseamos modificar el aspecto de un carácter al que puede accederse directamente a través del teclado.

Con respecto a 1): A veces es razonable introducir bucles de espera en un programa, e invitar al usuario con una frase en la pantalla a pulsar, por ejemplo, la tecla de entrada (señalada por ENTER o RETURN) para continuar la ejecución del programa. El problema que aparece al utilizar el comando INPUT$, es que no podemos leer la entrada directamente en la pantalla:

vea listado de ASC 1 del apéndice

Resultado: una línea vacía

En la pantalla no aparece nada al pulsar ENTER después de la entrada pedida. Sin embargo, si observamos la longitud del string A$, veremos que a pesar de ello se ha entrado un carácter, aunque no esté indicado en la pantalla:

```
PRINT LEN(A$)
```

resultado: 1

Utilicemos ahora el comando ASC para interpretar con más exactitud el carácter indicado:

vea listado de ASC 1a del apéndice

resultado: 13

La tecla ENTER tiene, pues, el valor 13 para el ordenador. Ahora podemos modificar nuestro programa de forma que su ejecución sólo continúe cuando se pulse dicha tecla (ENTER):

vea listado de ASC 1b del apéndice

Con respecto a 2): Como ya sabemos, el sistema MSX tiene un juego de caracteres ligeramente diferente de lo habitual. Si bien es verdad que la representación de los caracteres entre 32 (espacio) y 126 (línea ondulada) está normalizada, de 1 a 31 y de 127 a 254 el sistema MSX representa caracteres especiales.

Con ayuda de una impresora MSX, todos estos caracteres sin excepción, también pueden ser representados directamente mediante una salida LPRINT; sin embargo, no es así si conectamos otro tipo de impresora a la interfase CENTRONICS. Cualquier impresora emite cosas, pero al utilizar caracteres fuera del ámbito normalizado (de 32 a 126), se producirán sorpresas (cambio a otro tipo de letra, cambio a representaciones gráficas de alta resolución, etc.). Lo mejor será que usted mismo compruebe si su impresora es del tipo MSX u otra normal:

```
10 SCREEN ,,,,0
20 LPRINT " "     'pulsar teclas GRAPH y P
```

Si en su impresora aparece ahora el carácter de bloque

indicado, todo está en orden y podrá pasar a leer 'con respecto a 3)', puesto que posee una impresora MSX y no necesita arreglárselas con los problemas de una impresora no-MSX. En el caso contrario, en cada programa que deba emitir gráficos por la impresora, tendrá que entrar previamente el comando siguiente:

```
SCREEN ,,,,1
```

Con ello se indica al ordenador, que deberá emitir en la impresora conectada cualquier carácter gráfico en forma de espacio.

Sin embargo, ¿cómo puede comunicar dichos caracteres a un amigo, que también posee un ordenador MSX, o a una revista, a la que desea enviar sus propios programas MSX para imprimirlos? O indica espacios en lugar de los caracteres gráficos para imprimirlos y, a continuación, actúa de pequeño artista (dibujando los caracteres en el listado), o convierte los gráficos no representables por su impresora en su correspondiente carácter numérico, puesto que un carácter gráfico no es más que un simple número (vea arriba). Para ello deberá extraer cada carácter gráfico de un tabla (vea resumen del apéndice), o bien decodificarlo con ayuda del comando ASC. El siguiente programa corto podrá realizarlo con 255 caracteres seguidos:

vea listado de ASC 2 del apéndice

En lugar de los caracteres gráficos (que la impresora no-MSX no puede representar) puede entrar ahora en su listado:

```
LPRINT CHR$(180);CHR$(...);...
```

Como puede ver en el comando CHR$(), este comando convierte de nuevo los números en letras, cifras o caracteres gráficos.

Con respecto a 3): Debemos conocer el código ASC del carácter a convertir para modificar la dirección correcta de la VRAM (memoria RAM de vídeo). Por ejemplo, queremos modificar el aspecto del espacio. El cálculo de la posición de memoria, donde comienza la estructura del espacio, podrá realizarse con el siguiente programa:

vea listado de ASC 3 del apéndice

resultado: 2304

advertencia: en otro capítulo de este libro se habla más detenidamente de los comandos BASE y SCREEN.

El número 2304 indica la posición de la memoria VRAM donde comienza el almacenamiento del espacio (en total son ocho las posiciones que se encargan del aspecto del espacio: de 2304 a 2311). Entrando otro valor en la posición de memoria verá lo que sucede:

```
VPOKE 2304,255
```

Por lo visto, toda la pantalla ha cambiado de golpe, como se verá mejor al entrar CLS. Sin embargo, la súbita modificación de la pantalla sólo se debe a su composición formada casi exclusivamente de espacios. Esto significa que, al modificar el aspecto de un carácter -en este caso del espacio-, el ordenador cambia el aspecto de todos los caracteres del mismo tipo (con el mismo número ASC) en la pantalla, ¡incluso en el listado! También podemos anular este proceso con ayuda de un comando SCREEN (todos los cambios de la VRAM serán devueltos a sus valores estándar), o bien entrando otro POKE:

```
VPOKE 2304,0
```

Ahora todo vuelve a la normalidad, y espero que haya aprendido alguna cosa con el comando ASC.

Otras dos advertencias para aplicar el comando ASC de forma útil:

1) También al decodificar caracteres con ASC, el tamaño de la emisión se mueve dentro de los límites mágicos del ordenador. Esto significa que el ordenador emitirá siempre valores comprendidos entre 0 y 255, nunca inferiores ni superiores.

2) No utilice nunca el comando ASC con strings vacíos, por ejemplo:

```
10 A$=""
20 PRINT ASC(A$)
```

Si no hay nada, su ordenador no podrá calcular nada. El resultado será el mensaje de error 'illegal function call'.

ASC 1

```
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Pulse la tecla ENTER"
40 A$=INPUT$(1)
50 PRINT A$
```

ASC 1a

```
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Pulse la tecla ENTER"
40 A$=INPUT$(1)
50 PRINT ASC(A$)
```

ASC 1b

```
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Pulse la tecla ENTER"
40 A$=INPUT$(1)
50 IF ASC(A$)=13 THEN PRINT "Listo" ELSE
 GOTO 30
```

ASC 2

```
10 COLOR 1,15
20 SCREEN 0
30 INPUT "Escriba alguna cosa ";A$
40 FOR N=1 TO LEN(A$)
50 PRINT MID$(A$,N,1);" = ";ASC(MID$(A$,
N,1)),
60 NEXT N
```

ASC 3

```
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Espacio = CHR$(32)"
40 START=2048+32*8
50 PRINT START
```

Comandos Gráficos DRAW

Lenguaje de programación para dibujar líneas

vea también los programas del apéndice: DRAW 1
DRAW 2
DRAW 3
DRAW 4
DRAW 5
DRAW 6
DRAW 7
DRAW 8
DRAW 9
DRAW 10
DRAW 11
DRAW 12
DRAW 13
DRAW 14

Tratar DRAW como 'comando' sería menospreciarlo. Por otra parte, sería exagerado hablar de la utilización del comando DRAW como si fuera un segundo lenguaje de programación del ordenador MSX después del BASIC. Es cierto que detrás de DRAW se encuentran numerosos subcomandos, tal como demuestra la longitud de este capítulo. Estos subcomandos tienen un único objetivo: dibujar líneas. A pesar de ello, este espectro de comandos se parece muy poco a un verdadero lenguaje de programación de gráficos (tan sólo el subprograma LOGO gráficos de tortuga puede imitarse con el comando DRAW del ordenador MSX).

Después de menospreciar las posibilidades del comando DRAW (el fabricante del BASIC MSX, la firma americana de software MICROSOFT, con el comando DRAW se refiere al GML = Graphic Macro Language) vamos a analizarlo detenidamente:

vea listado de DRAW 1 del apéndice

Antes de detallar los comandos DRAW utilizados en este programa, quiero advertir que al utilizar dicho comando se está trabajando en el nivel gráfico. Sin embargo, los

gráficos sólo pueden representarse en SCREEN 2 y SCREEN 3, de lo contrario se emite el mensaje de error 'Illegal function call'.

Repasemos los caracteres detrás del comando DRAW paso a paso: las comillas nos indican que aquí se trata de un string (al igual que A$="Jose"). De momento no tiene importancia, pero más adelante ya apreciará las posibilidades del tratamiento de strings con el comando DRAW.

Al principio indicamos a nuestro ordenador que desplace el cursor gráfico hacia el centro de la pantalla (128,96), lo que se consigue precisamente con 'BM 128,98'; la 'B' significa BLIND, que quiere decir tanto como: no traces ninguna línea en la pantalla hasta llegar a las coordenadas 128,96. 'M' es la abreviación de MOVE, que significa: mueve el cursor gráfico desde su última posición hacia las coordenadas indicadas a continuación (128,96).

La escritura completa de 'B M 128,96' sería pues: BLIND MOVE TO 128,96 (movimiento sin dibujo del cursor hacia la coordenada 128,96).

Con algunos conocimientos del inglés, no tendrá dificultades en traducir las cuatro direcciones:

hacia arriba = Up
hacia abajo = Down
hacia la izquierda = Left
hacia la derecha = Right

Para ahorrar tiempo y memoria al efectuar la entrada, nuestro ordenador MSX se conforma con que le indiquemos la primera letra de la dirección en cuestión, o sea:

'U' para Up
'D' para Down
'L' para Left
'R' para Right

Además de indicar cada dirección, el ordenador MSX necesita otro dato para saber hasta dónde debe llegar el viaje. Es decir, que la entrada 'R8' significa simplemente: desplázate 8 puntos de pantalla hacia la derecha.

Aunque sólo hemos conocido unos pocos subcomandos DRAW, sabremos interpretar ahora nuestro ejemplo:

```
DRAW "B M 128,96 U8 R8 U8 R8 D16 L16"
```

Ve hacia la posición de coordenadas 128,96, sin trazar una línea llegando hasta ahí; mueve el cursor 8 puntos hacia arriba, después 8 puntos hacia la derecha, otra vez 8 puntos hacia arriba y 8 puntos hacia la derecha, 16 puntos en sentido descendente y 16 hacia la izquierda; ya ha trazado nuestro pequeño gráfico, es decir una escalera.

Respecto a la escritura diremos: en este libro hemos introducido espacios entre los diversos subcomandos de DRAW para facilitarle la lectura. Sin embargo, puede prescindir de ellos; también puede colocar punto y coma en lugar de los espacios utilizados para separar los comandos. Sin embargo, en un único punto es obligatorio colocar una coma: para expresar 2 coordenadas de un punto de la pantalla (X,Y) es imprescindible separar ambas mediante una coma.

Continuemos hablando de los subcomandos de DRAW. Además de los cuatro puntos cardinales, también podemos utilizar valores intermedios (en meteorología se habla por ejemplo de SE o sureste, NO o noroeste ...). Puesto que en el BASIC MSX, estas direcciones también pueden expresarse con una única letra, los creadores de estos subcomandos de DRAW han tenido una excelente idea: utilizar el alfabeto a partir de la letra 'E' para indicar los puntos cardinales de la forma siguiente:

Noreste = E
Sureste = F
Suroeste = G
Noroeste = H

Con ayuda de estas ocho direcciones podemos dibujar ahora una casa (¿recuerda el dibujo de la infancia? ¿Dibujar una casa sin levantar el lápiz o repetir una línea?):

Para ello traducimos un posible orden de ejecución al comando DRAW:

vea listado de DRAW 2 del apéndice

Sin embargo, nuestro comando DRAW del MSX esconde además muchas otras facultades. ¿Qué le parecería, por ejemplo, representar dicha casa de otro tamaño? Muy sencillo, dirá: si el tamaño debe doblarse, simplemente doblamos los números indicados tras el comando DRAW, de forma que la línea 30 queda como sigue:

```
30 DRAW "B M 128,96 R16 U16 L16 E8 F8 G16 U16 F16"
```

Esta forma es válida y funciona perfectamente, aunque los ingenieros de software del MSX opinan que es demasiado complicada. En su lugar utilizamos otra letra seguida de indicación numérica, para definir el factor de aumento o reducción de nuestro dibujo. Dicha letra es la 'S', que significa algo así como factor de escala. Para realizar un dibujo de tamaño normal (o sea, DRAW "R8", que desplaza el cursor 8 puntos hacia la derecha), debe utilizarse en factor de escala cuatro ("S4"). La representación doble (relación de escala = 2:1) requiere un valor de S 2*4 = 8 (DRAW "S8"). Si hay que reducirlo a la mitad, (relación de escala = 1:2), su valor debe ser 2 (la mitad del valor original: 4/2 = 2). Ampliaciones y reducciones pueden realizarse con valores entre 1 y 255 junto con la letra "S". El ordenador MSX también acepta valores que superan el margen de la pantalla (cosa que puede ocurrir fácilmente al utilizar valores de dos y tres cifras). En este caso, el dibujo que se encuentra

en un extremo fuera del borde de la pantalla, será mostrado junto a dicho borde, como por ejemplo en el programa siguiente:

vea listado de DRAW 3 del apéndice

Por favor, interrumpa la representación en la pantalla pulsando las teclas CTRL y STOP y ejecute el programa de nuevo con RUN. Verá que, en lugar del cuadrado antes definido, ahora se muestra otro de tamaño exagerado. ¿Cómo se explica eso? Al describir el comando COLOR ya mencioné que el ordenador MSX retiene muchos datos en la memoria, como por ejemplo los colores de primer plano, fondo y área de los bordes una vez definidos mediante COLOR. El factor de escala se comporta de forma parecida: si no es reactivado a tiempo (en este caso al tamaño normal = "S4"), también será conservado para otros comandos DRAW. Ello nos evita tener que definirlo nuevamente en cada línea del programa, si hay que utilizar varias veces el mismo factor. Esto significa que debemos modificar la línea 30 de nuestro programa si deseamos reejecutarlo continuamente en su estado original:

```
30 DRAW "B M 128,96 S4 R8 U8 L8 D8"
```

Otra letra que puede introducirse en el comando DRAW es la A, como ángulo. Ella permite girar los caracteres creados en ángulos de 90 grados:

vea listado de DRAW 4 del apéndice

No se ha girado nada todavía, para que pueda recordar el estado original del dibujo: una escalón que sube a partir del centro. Ahora vamos a cambiar la línea 30 con el nuevo comando A:

```
30 DRAW "B M 128,96 A1 R8 U8 R8"
```

Se mire como se mire ahora el dibujo: el escalón conduce hacia abajo. Los comandos de 'derecha' y 'arriba' han sido girados 90 grados en el sentido del reloj, de manera que la

función R (hacia la derecha) se ha convertido en una función D (hacia abajo). Igualmente sucede con la función U (hacia arriba): al girarla 90 grados en sentido del reloj se ha convertido en R (derecha). Es decir, podemos fijar las siguientes reglas:

A = 1 o sea	U = R (arriba = derecha)
	R = D (derecha = abajo)
+90 grados	D = L (abajo = izquierda)
	L = U (izquierda = arriba)

Si hubiésemos dibujado una torre, ésta se habría tumbado ahora sobre su lado derecho inicial.

Si aumentamos los valores de A del comando DRAW, el ángulo crece cada vez más (en el sentido del reloj), pero siempre en pasos de 90 grados:

A = 2 o sea	U = D (arriba = abajo)
	R = L (derecha = izquierda)
+180 grados	D = U (abajo = arriba)
	L = R (izquierda = derecha)

A = 3 o sea	U = L (arriba = izquierda)
	R = U (derecha = arriba)
+ 270 grados	D = R (abajo = derecha)
	L = D (izquierda = abajo)

Con ayuda de un pequeño subprograma incluso es posible girar nuestros dibujos también cualquier otro ángulo con el comando DRAW.

¿Para qué se utiliza el comando A de DRAW? Después de aprender su manejo es fácil dibujar, por ejemplo, una pequeña brújula. Simplemente hay que definir la forma de la flecha y dibujarla varias veces con ayuda de un factor de dirección redefinido cuatro veces:

vea listado de DRAW 5 del apéndice

Aunque la punta de la flecha deba volver a señalar siempre de nuevo hacia arriba (tal como se indica en la línea 30), el factor A de valor creciente en el comando DRAW determina finalmente su dirección. De esta forma se gira la dirección, y una flecha apunta hacia arriba (Norte), una hacia la derecha (Este), otra hacia abajo (Sur) y la última hacia la izquierda (Oeste), confeccionando así el aspecto de una brújula. Imagínese la extensión del programa si se hubiese tenido que girar (y efectuar los cálculos correspondientes) la dirección cada vez.

Al llegar a otros subcomandos de DRAW, se dará cuenta de lo fácil que es el trabajo con el comando DRAW, sabiendo cómo utilizarlo.

Puedo aconsejarle otra aplicación útil del comando A de DRAW: más adelante se muestra un listado que nos permite utilizar letras y cifras en las pantallas gráficas conayuda del comando DRAW. Naturalmente, esta letra de DRAW permitirá desplazarla a lo largo de una recta dibujada (vertical), o incluso titular de forma invertida un detalle de nuestro dibujo.

Antes de teclear este programa impresionante, le presentamos otro subcomando de DRAW: la letra N sirve para indicar al ordenador MSX que vuelva a su punto de partida después de terminar el dibujo encargado.

Para poder demostrar este comando en la práctica simplificamos la brújula confeccionada anteriormente borrando la punta de la flecha, con lo que el programa inicialmente largo se reduce a poco más de una línea:

vea listado de DRAW 6 del apéndice

Al principio del comando DRAW, el ordenador recibe la orden de comenzar el dibujo en la posición 128,96 (centro de la pantalla). La letra N indicada a continuación determina que la instrucción de dibujo deberá ser ejecutada, pero que después deberá volver al último punto mencionado del gráfico (en este caso el centro 128,96).

Tal como hemos explicado en el capítulo anterior (subcomando A de DRAW), el valor creciente de A indica que el comando U (hacia arriba) debe considerarse girado 90, 180 y finalmente 270 grados. De esta forma se crea pues la rosa de los vientos.

¿Desea un tamaño mayor? Si ha leído atentamente las explicaciones anteriores del comando DRAW, podrá aumentar el tamaño fácilmente. Ponga una 'S' de factor de escala delante de la propia rutina de dibujo y, a continuación, indique un número mayor que 4, por ejemplo:

```
30 DRAW"B M 128,96 S12 N A0 U8 N A1 U8 N A2 U8 N A3 U8"
```

Al principio del presente capítulo hemos hablado de los subcomandos M y B: la M seguida de dos coordenadas (separadas mediante una coma) indica que el cursor debe desplazarse hacia el punto señalado a partir de su última posición en la pantalla (M = to move = mover), dibujando, sin embargo, la línea del color graduado previamente.

Si deseamos que el movimiento del cursor no vaya acompañado de una línea hasta el destino, debemos colocar una B delante de la M, que significa: ve ciegamente hacia el punto de destino, es decir, no traces ninguna línea.

Puesto que al utilizar el comando DRAW "B M" el cursor se mueve sin efectuar ningún dibujo, este comando puede aplicarse -de forma parecida al comando LOCATE de las pantallas de texto- para efectuar el posicionamiento punto por punto (también de textos gráficos).

Sin embargo, también podemos utilizarlos de otra forma. Si

sabe manejar los comandos LINE, PSET y CIRCLE del MSX, seguramente conocerá la diferencia entre direccionamiento relativo (con STEP) y absoluto. Al manejar el direccionamiento absoluto debemos partir del comienzo de la pantalla, que se encuentra en la posición 0,0, o sea en la esquina superior izquierda (si deseamos realizar algún dibujo en el centro de la pantalla, debemos calcularlo (128,96) dentro de los límites disponibles de puntos (256 * 192), ordenando el inicio del dibujo en este punto). Dicho de forma sencilla: absoluto significa, que 0,0 es la esquina superior izquierda; 255,191 es la esquina inferior derecha, y 128,96 corresponde al centro de la pantalla. El direccionamiento relativo, en cambio, requiere la definición de un punto de partida nuevo, a partir del cual (STEP) deberá producirse el movimiento de tantos pasos hacia la derecha, arriba, la izquierda o abajo. El direccionamiento relativo tiene la ventaja, de que los dibujos ya confeccionados en un lugar de la pantalla,también pueden ser realizados de nuevo en otro lugar.

Mientras que el direccionamiento relativo en los principales comandos gráficos es indicado mediante STEP, dentro del comando DRAW es efectuado de forma más sencilla: al indicar las coordenadas utilizamos los valores numéricos habituales, pero añadiendo además un signo positivo (+) o negativo (-) para determinar el direccionamiento relativo. Cuando el ordenador lea esta instrucción, sabrá inmediatamente que el punto original debe ser considerado como nuevo punto de partida. El ejemplo siguiente dibujará simplemente cuatro cajitas en fila:

vea listado de DRAW 7 del apéndice

Sencillo y convincente, ¿verdad? Modifiquemos ahora la línea 50 de manera que las cajitas formen una diagonal:

```
50 DRAW "B M +8,+8 R4 U4 L4 D4"
```

Cuando comience a utilizar el direccionamiento relativo en estructuras más complicadas de dibujos, también sabrá

apreciar esta modalidad del comando gráfico DRAW (por ejemplo, el juego de caracteres de DRAW mencionado anteriormente, trabaja únicamente de esta forma en la pantalla gráfica, porque de lo contrario, sólo podríamos situar nuestra escritura en determinada posición absoluta de la pantalla, en lugar de colocar los caracteres uno al lado de otro a distancias regulares).

Otro ejemplo práctico de aplicación es la representación de figuras de ajedrez en un tablero: aquí se trata de dibujar 16 peones ... No hay nada más fácil que utilizar el direccionamiento relativo con ayuda del comando DARW "B M":

vea listado de DRAW 8 del apéndice

¿Hubo alguien que se burló de la designación de 'lenguaje de programación completo' al hablar del comando DRAW? Espero que ya se haya convencido por sus múltiples aplicaciones y que lo encuentre ahora un poco más fascinante.

Vayamos ahora al próximo subcomando: introducimos simplemente alguno de los 15 (o bien 16) colores en nuestro dibujo. Este proceso se realiza con la letra "C" (del inglés COLOUR = color) seguido de un número de una o dos cifras. Para volver al estado original después del ejemplo anterior, al principio del comando DRAW indicamos "S4 A0" para resetear el ordenador:

vea listado de DRAW 9 del apéndice

Después de volver a posicionar el cursor en el centro de la pantalla (128,96), "C15" indica que a partir de aquí se trazarán 16 puntos blancos hacia la derecha; a continuación se modificará el color antes de cada movimiento, de manera que finalmente se muestra un cuadrado enmarcado por dos barras negras ("C1") y dos blancas ("C15") en la pantalla.

Puede conseguirse el mismo efecto si entretanto se vuelve a utilizar contínuamente el comando COLOR. Sin embargo, aquí hay que actuar con precaución, de lo contrario también se

modificará simultáneamente el color de la escritura en las pantallas de texto (mientras que el comando COLOR influye en gráficos y textos, algunos subcomandos gráficos de color sólo hacen referencia a gráficos aislados). No obstante, observemos ahora el programa anterior después de añadir el comando COLOR

vea listado de DRAW 10 del apéndice

El programa ha adquirido una longitud considerable, aunque sólo efectúa lo mismo que el programa anterior, que utiliza el subcomando C de DRAW. Además faltaría añadir otra línea de la forma

COLOR 15

para volver al color inicial.

¡Otra advertencia! ¡No modifique nunca el color aleatoriamente en el ordenador MSX si las distancias entre puntos son demasiado pequeñas! Aunque admitan 256*192 puntos gráficos, la diferenciación entre colores se efectúa en una matriz de 32*192, para evitar que se superpongan como manchas de color en el siguiente ejemplo:

vea listado de DRAW 11 del apéndice

Al igual que en los ejemplos anteriores, este programa también debería mostrar dos líneas blancas y dos negras. Sin embargo, no es así por culpa de la mezcla de colores demasiado juntos (color de fondo más blanco más negro, es decir, que sobra al menos un color).

Hasta ahora hemos conocido todos los comandos, o sea letras, que integran el lenguaje de programación DRAW y que son:

R U L D	para las direcciones de 90 grados
E F G H	para las direcciones intermedias de 45 grados
A	para el factor de giro de 90 grados
M	para movimientos del cursor con dibujo

B M   para movimientos del cursor sin dibujo
C   para los cambios de color
S   para ampliaciones y reducciones
M 128,96   para el direccionamiento absoluto
M +50,-50 para el direccionamiento relativo

Para finalizar explicaremos otros dos comandos, que no son ampliaciones de DRAW en forma de subcomandos, pero pueden facilitarnos considerablemente la programación de gráficos con ayuda del comando DRAW.

1) Podemos introducir variables en los strings de DRAW, utilizando el siguiente formato en lugar de una entrada numérica:

= nombre de variable ;

Lo de nombre de variable parece claro; ¡pero no olvide nunca entrar además el signo de igualdad precedente ni el punto y coma que le sigue a continuación! De lo contrario, en caso de duda, el ordenador no sabrá lo que debe hacer; por ejemplo, al utilizar la variable A intentaría girar el dibujo en 90 grados. Por lo tanto, debe vigilar siempre la sintaxis correcta al efectuar la entrada, tal como muestra el ejemplo siguiente:

vea listado de DRAW 12 del apéndice

¡Caramba! En algo menos de dos segundos, su ordenador MSX le ha mostrado una multitud de cuadrados de tamaño creciente en la pantalla. Mientras que el dibujo inferior tiene el tamaño original, el factor de ampliación ha crecido continuamente hasta llegar a mostrarlo todo ampliado veinticinco veces con "S=100" (es decir, 25*4 = en lugar de 4 puntos se han dibujado 100 puntos; lo que ha sucedido entre el cuadrado menor y el superior puede observarse sucesivamente en la pantalla).

La asignación de variables mediante "=" y ";" puede utilizarse en todos los subcomandos DRAW, como por ejemplo,

al dibujar la rosa de los vientos utilizando el comando angular con "A":

vea listado de DRAW 13 del apéndice

Tal como habrá notado seguramente en el último programa, al final (en este caso la línea 60 del programa) devolvemos los parámetros modificados a su estado habitual. De esta forma no pueden aparecer complicaciones al ejecutar de nuevo el programa. En realidad, el estado normal completo es el siguiente:

A0 (ángulo = 0 grados)
C15 (color blanco de primer plano)
S4 (sin ampliaciones ni reducciones)
B M 0,0 (cursor gráfico en el punto de salida, es decir, en la esquina superior izquierda)

2) Podemos introducir cadenas de caracteres de DRAW ya confeccionadas en otra cadena de caracteres de DRAW en forma de subcadenas.

Para demostrarlo no comenzamos dibujando directamente; en primer lugar confeccionamos una tabla de caracteres:

En la variable A$ almacenamos nuestra escalera del principio ("R8 U8 R8"),

en la variable B$, el cuadrado ("R8 U8 L8 D8") y finalmente

en la variable C$ almacenamos la casita ("R8 U8 L8 E4 F4 G8 F8").

La sintaxis para indentar las variables de cadena se inicia con la letra "X" y se finaliza nuevamente con un punto y coma:

X variable de cadena ;

Para indentar los tres strings se siguen los pasos que se muestran en el ejemplo:

vea listado de DRAW 14 del apéndice

El resultado no es precisamente espectacular, pero quizás al indentar las piezas dibujadas anteriormente, le demuestre al menos la capacidad y variedad que permite utilizarse al encadenar los diversos strings parciales de DRAW.

Tal como mencionamos al principio del presente capítulo, le recordamos nuevamente que el manejo de los strings de DRAW es igualmente sencillo que el de los strings normales (A$="Javier"). Por lo tanto, deberá tener en cuenta que un string no puede superar los 255 caracteres en un ordenador MSX. Además, en realidad debería reservar memoria previamente con ayuda del comando CLEAR (por ejemplo, para cuatro strings de 200 caracteres de longitud cada uno, en todo caso deberá entrar previamente: CLEAR 800).

Algunas indicaciones finales:

1) Los números que definen la dirección pueden tener valores entre +32767 y -32768; no es necesario que se limiten a los puntos que pueden representarse en la pantalla (256*192 puntos). Sin embargo, no se sabe para qué servirán estos valores tan elevados (posiblemente el fabricante del MSX quería contribuir a evitar la emisión de numerosos mensajes de error (y con ello el inevitable borrado de la pantalla)).

2) Se dispone de cuatro posiciones de memoria para leer el actual estado del cursor, o bien para entrar otro nuevo (lo último corresponde a DRAW "B M X Y"):

&HFCB3	cursor gráfico X absoluto
&HFCB5	cursor gráfico Y absoluto
&HFCB7	cursor gráfico X relativo
&HFCB9	cursor gráfico Y relativo

3) En la posición de memoria

&HFCBC

se memoriza el actual factor de escala de DRAW, o sea, el factor de ampliación/reducción, que también puede modificarse en dicha posición.

4) En la posición de memoria

&FCBD

se almacena el actual ángulo de DRAW, que también puede ser modificado en ella.

5) Finalmente, en la posición

&HF3E9

de la memoria se memoriza el actual color del primer plano, que podrá modificarse igualmente.

DRAW 1

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 U8 R8 U8 R8 D16 L16"
40 GOTO 40
```

DRAW 2

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 R8 U8 L8 E4 F4 G8 U8
 F8"
40 GOTO 40
```

DRAW 3

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 R8 U8 L8 D8"
40 DRAW "B M 128,96 S80 R8 U8 L8 D8"
50 GOTO 50
```

DRAW 4

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 R8 U8 R8"
40 GOTO 40
```

DRAW 5

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 U16 F4 H4 G4"
40 N=N+1
50 ON N GOSUB 70,90,110
60 GOTO 60
70 DRAW "A1"
80 GOTO 30
90 DRAW "A2"
100 GOTO 30
110 DRAW "A3"
120 GOTO 30
```

DRAW 6

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 N A0 U8 N A1 U8 N A2
 U8 N A3 U8"
40 GOTO 40
```

DRAW 7

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96"
40 FOR N=1 TO 4
50 DRAW "B M +0,+8 R4 U4 L4 D4"
60 NEXT N
70 GOTO 70
```

DRAW 8

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "S4 A0 B M 50,130"
40 FOR N=1 TO 8
50 DRAW "B M +18,+0 R8 U4 L3 U12 R3 U2 L
4 E3 H3 G3 F3 L4 D2 R3 D12 L3 D4
60 NEXT N
70 DRAW "S4 A0 B M 50,66"
80 FOR N=1 TO 8
90 DRAW "B M +16,+2 R8 U4 L3 U12 R3 U2 L
4 E3 H3 G3 F3 L4 D2 R3 D12 L3 D4
100 PAINT STEP(+2,-2)
110 NEXT N
120 GOTO 120
```

DRAW 9

```
10 COLOR 1,4
20 SCREEN 2
30 DRAW "S4 A0"
40 DRAW "B M 128,96 C15 R16 C1 U16 C15 L
16 C1 D16"
50 GOTO 50
```

DRAW 10

```
10 COLOR 1,4
20 SCREEN 2
30 DRAW "S4 A0"
40 DRAW "B M 128,96"
50 COLOR 15
60 DRAW "R 16"
70 COLOR 1
80 DRAW "U 16"
90 COLOR 15
100 DRAW "L 16"
110 COLOR 1
120 DRAW "D 16"
130 GOTO 130
```

DRAW 11

```
10 COLOR 1,4
20 SCREEN 2
30 DRAW "B M 128,96 C15 R4 C1 U4 C15 L4
C1 D4"
40 GOTO 40
```

DRAW 12

```
10 COLOR 1,4
20 SCREEN 2
30 FOR N=4 TO 100 STEP 4
40 DRAW "B M 128,96 S=N;R4 U4 L4 D4"
50 NEXT N
60 DRAW "S4"
70 GOTO 70
```

DRAW 13

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=0 TO 3
40 DRAW "B M 128,96 A=N;U8"
50 NEXT N
60 DRAW "A0"
70 GOTO 70
```

DRAW 14

```
10 COLOR 1,15
20 SCREEN 2
30 A$="S4 R8 U8 R8"
40 B$="R8 U8 L8 D8"
50 C$="S8 R8 U8 L8 E4 F4 G8 U8 F8"
60 DRAW "B M 128,96 X A$;X B$;X C$;"
70 GOTO 70
```

Comandos Gráficos PSET

Marca un punto de color en las pantallas gráficas

vea también los programas del apéndice: POS 6
PRESET 1

PSET es uno de los comandos gráficos que sólo ejercen una función razonable en SCREEN 2 y SCREEN 3. Si intentásemos utilizar PSET en las pantallas de texto SCREEN 0 y 1, se emitiría el mensaje de error 'Illegal function call'.

El formato de la instrucción PSET es:

```
PSET(X,Y),Z
```

donde X e Y son las coordenadas del punto a dibujar, mientras que Z indica su color (vea más adelante). Al igual que en los demás comandos gráficos, PSET ofrece la posibilidad de direccionar el punto en cuestión (indicado por las coordenadas X e Y) de forma absoluta o relativa. Absoluto significa que las coordenadas se refieren directamente a la matriz de puntos disponible de 256*192 puntos gráficos (por ejemplo, las coordenadas 128,96 indican que el punto se dibujará en el centro de la pantalla).

Para efectuar un direccionamiento relativo debe colocarse el término STEP delante de las coordenadas X-Y, que significa: desplázate a partir del último punto gráfico mencionado según las indicaciones de X,Y hacia la derecha/izquierda (X) y hacia arriba/abajo (Y). Observe el ejemplo siguiente:

PSET(128,96)
marca un punto en la posición de coordenadas 128,96

PSET STEP(10,10)
marca un punto a partir de la última posición 128,96
10 puntos hacia la derecha (138) y
10 puntos hacia abajo (106)

La indicación Z del comando PSET permite escoger un color

para el punto. En caso de omisión, el ordenador MSX utiliza el actual color del primer plano para dibujar el punto.

También es posible utilizar el comando PSET de forma que trabaje igual que el comando PRESET. Para ello debemos conocer la diferencia entre ambos comandos: mientras que PSET utiliza el actual color de primer plano en caso de omitir la indicación correspondiente, el comando PRESET utiliza el color del fondo. Puesto que gran parte de la pantalla gráfica suele estar dominada por el color del fondo, la utilización de PRESET equivale a un borrado puntual de la imagen. Para obtener el mismo resultado con el comando PSET, debe indicarse el color del fondo para marcar el punto. El número correspondiente al color en cuestión no se halla mediante un comando gráfico; hay que leer la posición de memoria &HF3EA, que dispone de la información requerida:

```
PSET(X,Y),PEEK(&HF3EA)=PRESET(X,Y)
```

Otra forma de utilizar el PSET en lugar de PRESET sin necesidad de leer en la memoria, consiste en memorizar el último color de fondo en una variable. Si ello se efectúa siempre, la variable sirve para añadirla al comando PSET en función de indicar el color del punto.

No está obligado a dibujar puntos sólo en la superficie representable de la pantalla (256*192 puntos), sino que puede indicar valores entre -32768 y +32767 para X e Y. Sin embargo, ello no tiene ningún sentido. Si se indican valores todavía menores o mayores que los señalados, se emite el mensaje de error 'Overflow'.

Al igual que en los demás comandos gráficos, el marcar un punto con PSET provoca una actualización del cursor gráfico. En cualquier momento puede llamar las coordenadas actuales (y también modificarlas mediante POKE), si se fija más detenidamente en las posiciones de memoria &HFCB3, &HFCB5, &HFCB7 y &HFCB9:

posición &HFCB3: coordenada X absoluta
posición &HFCB5: coordenada Y absoluta
posición &HFCB7: coordenada X relativa
posición &HFCB9: coordenada Y relativa

El cursor gráfico actual es memorizado en las posiciones &HFCB3 y &HFCB5, mientras que las posiciones &HFCB7 y &HFCB9 indican en cada momento su distancia hasta el cursor real (Ejemplo: dibujo de una circunferencia indicando el punto central, STEP utilizado en el comando PSET).

vea listado de POS 6 del apéndice

Otras dos advertencias:

1) Si desea actualizar sólo el cursor gráfico (parecido al comando LOCATE de las pantallas de texto), el comando

```
PRESET(X,Y) o
PSET(X,Y),color de fondo
```

constituye la mejor y más segura forma de hacerlo. La entrada del comando PSET sin indicación del color no sería muy razonable, puesto que la consecuencia sería disponer en el lugar del destino de un punto indeseado de pantalla. Por el contrario, PRESET marca un punto inicial invisible que coincide con el color del fondo.

vea listado de PRESET 1 del apéndice

Es cierto que su ordenador MSX dispone de un modo gráfico de alta resolución de 256*192 puntos, pero el gráfico de color correspondiente es muy deficiente: sólo dispone de 32*192 puntos de color en la pantalla (bloques horizontales). A causa de ello, al representar un gráfico hay que procurar que no coincidan dos colores diferentes de primer plano en un campo de 8*1 puntos, para evitar la superposición de colores (los colores se mezclan entre sí aparentemente de forma incontrolada).

Comandos Gráficos PRESET

Borra un punto de las pantallas gráficas

vea también los programas del apéndice: PRESET 1
PRESET 2

Si deseamos dibujar un punto en la pantalla en el modo gráfico SCREEN 2 ó SCREEN 3, nos servimos del comando PSET. La indicación de las coordenadas X-Y así como el número del color (0 a 15) sirven para definir dicho punto.

Sin embargo, a veces es interesante volver a borrar uno o varios puntos de la pantalla gráfica. En este caso debemos conocer el significado preciso de 'borrar': o activamos los puntos a borrar con el color del fondo de la pantalla, o los repintamos con un nuevo color de primer plano.

La aplicación del comando PRESET admite ambas posibilidades para borrar puntos:

PRESET(X,Y) = borrar el punto X,Y de la pantalla
PRESET(X,Y),Z = repintar el punto X,Y de la pantalla utilizando el color Z

Observemos la función de 'goma de borrar' del comando PRESET en el siguiente programa:

vea listado de PRESET 1 del apéndice

En primer lugar ha pintado (líneas 30 a 60) un cuadrado negro (color 1) sobre un fondo blanco (color 15). A continuación (líneas 70 a 110), el cuadrado negro ha sido repintado con el color del fondo (blanco = 15).

Utilizando el comando PSET junto con una indicación de color (que corresponde al color del fondo) se consigue el mismo efecto que con el comando PRESET. Es decir, que puede sustituir la instrucción de la línea 90 por:

```
90 PSET(M,N),15
```

De la misma manera, también puede sustituirse el comando PSET de la línea 50 por una instrucción PRESET correspondiente, indicando además el color que equivale al color escogido del primer plano:

```
50 PRESET(M,N),1
```

En realidad, los comandos PSET y PRESET cumplen pues las mismas funciones. La única ventaja de los dos comandos diferentes consiste en poder prescindir de la indicación del color en ambos (color de primer plano en PSET, color de fondo en PRESET), mientras que la instrucción inversa la exige forzosamente.

Para mantener un solo comando (o PSET, o PRESET) podemos orientarnos por las dos posiciones de memoria siguientes:

en &HF3E9 se almacena el color de primer plano
en &HF3EA se almacena el color de fondo

Con ayuda de ambas posiciones pueden sustituirse PSET y PRESET de la forma siguiente:

```
PSET(X,Y) = PRESET(X,Y),PEEK(&HF3E9)
PRESET(X,Y) = PSET(X,Y),PEEK(&HF3EA)
```

Podemos dirigirnos a los puntos de las pantallas gráficas (SCREEN 2 y 3) mediante los direccionamientos absoluto y relativo. Mientras que la entrada absoluta menciona las coordenadas directamente (por ejemplo, 128,96 corresponde al centro de la pantalla), el direccionamiento relativo parte de la situación del cursor gráfico en ese momento, aceptándolo como punto de partida (punto cero). El avance del movimiento se indica con valores de X e Y positivos o negativos. Para diferenciar el direccionamiento absoluto del relativo se utiliza el término STEP en este último:

```
PRESET STEP(10,-10)
```

Esto significa, que el próximo punto será colocado 10 puntos hacia la derecha (10) y 10 puntos hacia arriba (-10), contando a partir de la posición actual del cursor, y borrado allí. En este caso, el color utilizado para borrarlo es el actual color del fondo; con PSET debería utilizarse el color del primer plano.

Las entradas del comando PRESET no se limitan sólo a las coordenadas visibles en la pantalla, sino que pueden indicarse valores entre -32768 y +32767. Valores que exceden estos límites provocan un 'Overflow error'.

Los colores admitidos pueden tener valores entre 0 y 15; el color 0 equivale a transparente y no provoca cambios visibles en la pantalla. Al utilizar valores mayores que 15 y menores que 0, se emite el mensaje de error 'Illegal function call'.

Cualquier entrada PRESET actualiza el cursor grafico, de manera que

```
PRESET(128,96)
CIRCLE STEP(0,0),30
```

significa, que el centro de la circunferencia tiene las coordenadas 128,96.

El comando PRESET permite, gracias a su modificación invisible del cursor gráfico (parecido a DRAW"B M X,Y"), determinar las coordenadas iniciales para emitir texto en la pantalla gráfica:

vea listado de PRESET 2 del apéndice

En este aspecto, al comando PRESET le corresponde una función parecida a la del comando LOCATE de las pantallas de texto.

PRESET 1

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=100 TO 120
40 FOR M=80 TO 100
50 PSET(M,N)
60 NEXT M,N
70 FOR N=100 TO 120
80 FOR M=80 TO 100
90 PRESET(M,N)
100 NEXT M
110 NEXT N
120 GOTO 120
```

PRESET 2

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 PRESET(128,96)
50 PRINT #1,"MSX";
60 CLOSE
70 GOTO 70
```

Comandos Gráficos POINT

Lee cada uno de los puntos gráficos

vea también los programas del apéndice: POINT 1
POINT 2
POINT 3
programa 15

Cuando trabajamos en el modo gráfico de alta resolución SCREEN 2 ó en el de baja resolución SCREEN 3, nos hallamos en un sistema de coordenadas de 256*192 ó de 64*48 puntos.

En SCREEN 3, cada punto (o mejor dicho bloque) puede tomar cualquiera de los 16 colores disponibles (incluído el "color" transparente), pero no así en SCREEN 2.

SCREEN 2 ofrece un gráfico de alta resolución de 256*192 puntos, o sea, que podemos dirigirnos a casi 50000 puntos diferentes para representar gráficos en la pantalla; pero su resolución de color es muy deficiente: en sentido vertical se pueden cambiar los colores punto por punto sin provocar superposiciones; sin embargo, en sentido horizontal ello sólo es posible en secuencias de 8 puntos (puntos 0 a 7 = 2 colores diferentes, puntos 8 a 15 = 2 colores diferentes, etc.). Si intentamos mezclar un tercer color, éste se convertirá en color dominante de primer plano o de fondo, lo que afecta también a los puntos ya existentes de otros colores.

El comando POINT ofrece la posibilidad de inspeccionar el color de la pantalla punto por punto. Para ello basta indicar las coordenadas X-Y correspondientes y consultar una tabla de colores para decodificar el valor hallado.

vea listado de POINT 1 del apéndice

Puesto que la línea 30 dibuja un rectángulo de color negro (=1), cada punto de su superficie y de los bordes es negro.

En el siguiente programa, la variable dimensionada A lee el contenido de la pantalla gráfica y, a continuación, se muestra en la pantalla de texto la imagen gráfica en forma de códigos de colores:

vea listado de POINT 2 del apéndice

La línea 50 proyecta un pequeño rectángulo del tamaño de 8*8=64 campos en la pantalla. En las líneas siguientes, la variable dimensionada A lee la información de la pantalla gráfica punto por punto y, a continuación, sus valores demostrarán uno por uno en la pantalla de texto -pero esta vez en forma de códigos de colores- que la superficie leída corresponde a nuestro rectángulo. Mientras que el rectángulo ha sido pintado y rellenado con color 2, sus bordes son de color 3 (tal como define el comando COLOR de la línea 10).

Con ayuda de unas pequeñas modificaciones del programa anterior, también pueden leerse punto por punto las formas de los caracteres en la pantalla gráfica y mostrarlas ampliadas en la pantalla de texto. Observemos a tal fin el aspecto punto por punto de la letra "A":

vea listado de POINT 3 del apéndice

Puesto que el color del primer plano de la letra "A" es negro, los 1 emitidos muestran la forma de la "A".

También pueden indicarse otras letras o caracteres en la línea 70. Con un poco de fantasía, incluso puede ahorrarse la ayuda del generador de caracteres (vea programa 14) incluído en este libro. Con ayuda del programa anterior también puede conseguir efectos similares.

¿Qué aplicaciones tiene el comando POINT además de servir para los pequeños juegos que acabamos de practicar?

1) Para sacar una pantalla de alta resolución en una impresora de aguja o de matriz conectada al ordenador hay que transmitir punto por punto, tarea ideal que puede

ejecutarse fácilmente con el comando POINT:

vea programa 15 del apéndice

Sin embargo, al escribir una rutina de hardcopy, recuerde que sólo deben emitirse colores determinados en la impresora, por ejemplo mediante el comando:

```
IF POINT(X,Y)=3 THEN LPRINT ...
```

emite color en la impresora únicamente cuando el color de la pantalla es 3, ya que la mayoría de impresoras de matriz tienen sólo dos opciones: imprimir de color negro o dejarlo blanco, es decir, que debemos transformar una imagen en color en otra en blanco y negro.

2) Un juego a veces requiere que el programa lea informaciones de la pantalla, por ejemplo, un hombrecito camina sobre un muro rojo y debe saltarse agujeros. Puesto que estos agujeros seguramente no serán 'rojos', el comando POINT puede comprobar si los pasos del hombrecito son firmes o no (debajo de sus pies hay color rojo).

Otras advertencias para usar razonablemente el comando POINT:

1) POINT sirve para leer el color de los puntos de la pantalla al escoger SCREEN 2 y SCREEN 3. También en SCREEN 0 y SCREEN 1 se emiten valores, pero éstos no pueden aplicarse de forma útil, por ejemplo, para leer el gráfico confeccionado previamente en SCREEN 2. En este caso se trata más bien de leer bit por bit una zona de la memoria que no tiene ningún significado en la representación en pantalla de SCREEN 0 y SCREEN 1.

2) POINT 'sólo' permite leer (razonablemente) la ocupación de color de los 256*192 puntos de la pantalla gráfica. Sin embargo, también es posible indicar valores entre -32768 y +32767 (valores menores o mayores producen un error de 'Overflow') en lugar de las coordenadas X e Y del comando

POINT. En estos casos, la lectura del comando POINT da el valor -1.

3) El comando POINT puede ser sustituído por una lectura directa en la VRAM a través del comando VPEEK del BASIC. Sin embargo, para ello se requiere un conocimiento exacto sobre la configuración de la VRAM (vea en SCREEN, BASE y VDP), así como un programa bastante complejo de decodificación de la memoria, al menos en lo que se refiere a SCREEN 2: en este caso, la información de color se compone de datos procedentes de las tablas de colores y nombres de patrones.

4) Finalmente, una última aplicación del comando POINT: para confeccionar una propia rutina de comprobación de borde PAINT, se puede leer el correspondiente color del borde con POINT y comenzar después a pintar la superficie con PAINT sin ningún temor.

POINT 1

```
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(148,76),1,BF
40 A=POINT(138,85)
50 FOR N=1 TO 1000
60 NEXT N
70 SCREEN 0
80 PRINT A
```

POINT 2

```
10 COLOR 1,3
20 SCREEN 2
30 WIDTH 37
40 DIM A(12,12)
50 LINE(128,96)-(135,103),2,BF
60 FOR N=127 TO 136
70 FOR M=95 TO 106
80 A(N-126,M-94)=POINT(N,M)
90 NEXT M
100 NEXT N
110 SCREEN 0
120 FOR N=1 TO 10
130 FOR M=1 TO 10
140 PRINT A(M,N);
150 NEXT M
151 PRINT
160 NEXT N
```

POINT 3

```
10 COLOR 1,3
20 SCREEN 2
30 WIDTH 37
40 DIM A(10,10)
50 OPEN "GRP:" FOR OUTPUT AS #1
60 DRAW "BM 127,97"
70 PRINT #1,"A"
80 FOR N=127 TO 136
90 FOR M=97 TO 106
100 A(N-126,M-96)=POINT(N,M)
110 NEXT M
120 NEXT N
130 SCREEN 0
140 FOR N=1 TO 10
150 FOR M=1 TO 10
160 PRINT A(M,N);
170 NEXT M
180 PRINT
190 NEXT N
200 CLOSE
```

Comandos Gráficos LINE

Dibuja líneas y cuadrados en gráficos de alta resolución

vea también los programas del apéndice: LINE 1
LINE 2
LINE 3
LINE 4
LINE 5
LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11
LINE 12
LINE 13
DRAW 6

El comando gráfico LINE del MSX tiene significado triple:

1) Permite dibujar líneas
2) Permite pintar cuadrados
3) Permite pintar cuadrados y rellenarlos con color

Con respecto a 1): Por un lado permite indicar los cursores inicial y final, y por otro lado también permite tomar el cursor inicial a partir de su última posición y determinar simplemente la línea hacia la cual deberá dirigirse.

Al utilizar la última opción, hay que tener la precaución de que en su estado inicial (después de conectarlo), el ordenador MSX se encuentre en la esquina superior izquierda, en la posición 0,0. Observemos ambas opciones en el siguiente programa:

vea listado de LINE 1 del apéndice

En la línea 30 se da la instrucción de comenzar la línea a dibujar en el centro de la pantalla con posición del cursor 128,96 y de continuarla hasta el punto de coordenadas 150,150. Puesto que la línea 40 siguiente no indica ningún comienzo de línea, el ordenador MSX toma simplemente la última posición indicada como cursor inicial (150,150) y, a partir de ahí, continúa el trazado de la línea hasta la posición 160,120. Si cambiamos la línea 30 de la siguiente forma:

```
30 LINE-(150,150)
```

y reejecutamos el programa, el ordenador traza la línea a partir de la última posición del cursor indicada en la ejecución anterior del programa (160,120) hasta 150,150 (vea línea 30).

Sin embargo, para volver al estado inicial debemos añadir otra línea que permita al cursor gráfico comenzar el dibujo de nuevo en 0,0 (al igual que tras la conexión del ordenador):

```
25 DRAW "B M 0,0"
```

Ahora el cursor gráfico empieza en el punto 0,0 y traza una línea hasta el punto (indicado en la línea 30) 150,150.

Esto significa que antes de comenzar el dibujo,debe recordar siempre la posición actual del cursor gráfico, para evitar superposiciones inesperadas (sencillamente porque el cursor ha comenzado a dibujar en un punto erróneo de la pantalla) durante la ejecución del programa.

Además del direccionamiento absoluto (dentro del ámbito posible de 256*192 puntos de pantalla, se indican las coordenadas exactas de los puntos inicial y final del dibujo a realizar), el comando LINRE también admite la utilización del direccionamiento relativo con STEP:

vea listado de LINE 2 del apéndice

Aunque este dibujo ha sido creado más bien aleatoriamente, a continuación explicaremos línea por línea:

Línea 40: activado de forma absoluta, se dibuja una línea desde el centro de la pantalla (128,96) hasta el punto de coordenadas 150,150.

Línea 50: nos hallamos en la posición 150,150. El comando STEP precedente indica ahora, que el cursor gráfico debe moverse a partir de esa posición 10 puntos hacia la derecha y 10 puntos hacia abajo, antes de comenzar el dibujo. Por otra parte, a partir de ahí la línea se dirige de forma absoluta (sin STEP) hacia el punto de coordenadas 180,180.

Línea 60: puesto que aquí los puntos inicial y final de la nueva línea van precedidos de la indicación STEP, en este caso se procede sólo de forma relativa: el principio de la línea (-20,-20) se encuentra desplazado 20 puntos hacia la izquierda y 20 hacia arriba, contando a partir de la última posición del cursor. A partir de aquí, la línea se desplaza -40,-40, es decir, 40 puntos hacia la izquierda y 40 hacia arriba.

Puesto que los ordenadores MSX comienzan su dibujo - de forma inhabitual para los matemáticos- en la posición superior izquierda (posición del cursor 0,0), es conveniente traducir más exactamente las direcciones positivas y negativas del direccionamiento relativo por razones de claridad:

```
LINE STEP(x1,y1)-(x2,y2)
```

Si x1 es positivo (p.e. 3,3) - hacia la derecha
Si x1 es negativo (p.e. -3,3) - hacia la izquierda
Si y1 es positivo (p.e. 3,3) - hacia abajo
Si y1 es negativo (p.e. 3,-3) - hacia arriba

Si el sistema operativo procediese correctamente desde el punto de vista matemático-geométrico (orientado según el

sistema de coordenadas), el punto 0,0 de la pantalla debería hallarse en la esquina inferior izquierda. Por otro lado, el punto extremo (255,191) suele encontrarase en la esquina superior derecha y no en la inferior derecha tal como sucede en los ordenadores MSX. De esta forma, siguiendo una orientación ilógica matemáticamente hablando, debemos guiarnos por la tabla antes descrita.

El comando LINE, tal como se conoce hasta ahora, ya permite diseñar por ejemplo dibujos de figuras, ...

Ventaja respecto al comando DRAW: no estamos limitados a trazar líneas en sólo en las ocho direcciones señaladas, sino que podemos movernos en cualquier ángulo. ¡Probémoslo!

<u>vea listado de DRAW 6 del apéndice</u>

Si ya ha leído el capítulo sobre el comando DRAW, recordará la rosa de los vientos confeccionada en este programa.

Con ayuda del comando LINE podemos escoger cualquier ángulo, por lo que el programa siguiente nos muestra toda una superficie pintada:

<u>vea listado de LINE 3 del apéndice</u>

Se dirige sucesivamente (línea 30) a todas las direcciones posibles, de forma que crea finalmente un triángulo coloreado.

El siguiente programa demuestra que el comando LINE también puede dibujar radios a distancias superiores entre sí (parecido al comando DRAW de 45 grados):

<u>vea listado de LINE 4 del apéndice</u>

Este programa dibuja radios uno tras otro, hasta completar finalmente una superficie continua (en pasos de STEP 1, al igual que en el programa anterior).

Con respecto a 2): El comando LINE no sólo permite trazar una línea de x1,y1 a x2,y2, sino también utilizar los puntos inicial y final para circunscribir un rectángulo.

Observemos en el siguiente programa la forma de dibujar un rectángulo utilizando el comando LINE normal:

vea listado de LINE 5 del apéndice

Este programa utiliza el direccionamiento absoluto (tal como sabemos dibujar actualmente un rectángulo con el comando LINE). Al observar los valores indicados en las líneas 30 a 60, veremos que el programa sólo ha precisado 2 valores X y 2 valores Y para dirigirse a los cuatro puntos diferentes:

x1=128 x2=148
y1= 96 y2= 76

Veamos qué línea se visualizará en la pantalla al utilizar todas estas coordenadas en un único comando LINE:

vea listado de LINE 6 del apéndice

Es la diagonal que une las esquinas inferior izquierda y superior derecha del rectángulo dibujado anteriormente.

A partir de estos dos puntos definidos (128,96 y 148,76) el ordenador MSX es capaz de calcular las coordenadas de las dos esquinas restantes del rectángulo a dibujar:

Ya existen las coordenadas:
128,96 148,76

De ellos se hallarán las coordenadas:
128,76 148,96

Por supuesto, el ordenador las ordena automáticamente de tal forma que se origina finalmente un rectángulo:

1)	128,96		128,96
2)	128,76	ó	148,96
3)	148,76		148,76
4)	148,96		128,76

Si el orden de las cuatro coordenadas no fuese correcto, el rectángulo en cuestión estaría formado parcialmente por diagonales y lados.

En el idioma inglés, un rectángulo también se designa con "box". Por esta razón, los creadores del BASIC MSX han tomado (al igual que en los comandos DRAW y PLAY) simplemente la letra inicial de dicho término para ampliar el comando LINE; así que para dibujar un rectángulo o "box" se utiliza el comando:

```
LINE(x1,y1)-(x2,y2),,B
```

o bien, conocido el punto inicial:

```
LINE-(x2,y2),,B
```

Por favor, no olvide indicar dos comas delante de la letra "B". Más adelante explicaremos la razón de ello.

El siguiente programa muestra la rapidez con que el ordenador MSX puede dibujar gran cantidad de cuadrados o rectángulos en la pantalla:

vea listado de LINE 7 del apéndice

¡Es impresionante con qué rapidez ha dibujado 1000 rectángulos!

No queremos insistir más en el tema; tan sólo mencionar que también pueden dibujarse rectángulos utilizando el comando STEP (explicado exhaustivamente en la parte 'con respecto a

1)') para el direccionamiento relativo (el cursor gráfico actual es considerado como origen 0,0), tal como se muestra en el ejemplo siguiente:

<u>vea listado de LINE 8 del apéndice</u>

De esta forma se aclaran dos cosas:

1) Basta con indicar las diagonales (de la esquina inferior izquierda a la superior derecha) para dibujar un rectángulo (en este caso 10 rectángulos).

2) Estos diez pequeños rectángulos vuelven a definir finalmente la diagonal mencionada (pero a mayor escala) en la pantalla (provocado por el direccionamiento relativo mediante STEP).

También podemos conseguir el mismo efecto al [illegible] un lado en el comando LINE (indicando sólo X2 e Y2):

<u>vea listado de LINE 9 del apéndice</u>

Con ayuda de estos programas puede probar su capacidad de creación, por ejemplo, para representar un retículo formado por múltiples rectángulos, en lugar de trabajar en el modo de alta resolución, etc.

Con respecto a 3): Tras la "B" para indicar box (vea 'con respecto a 2)'), podemos añadir además otra letra: la F, para indicar que el rectángulo en cuestión debe ser colorido (del mismo color que el utilizado para los lados del rectángulo).

Observemos este proceso en el programa anterior que dibujó los 1000 rectángulos:

<u>vea listado de LINE 10 del apéndice</u>

Al principio es muy impresionante, pero al cabo de poco

tiempo parece que ya no sucede nada en la pantalla. Ello se debe a que los nuevos rectángulos creados van ocultando los existentes ... es decir: si ya existe un dibujo en el lugar donde aparece otro nuevo, el ordenador MSX no visualiza este último, considerando dicho lugar ocupado por el anterior. Ni el ordenador MSX ni las pantallas de TV son capaces todavía de desarrollar un efecto tridimensional. Para demostrar que continuamente se van creando nuevos rectángulos, podemos proceder de dos maneras diferentes:

1) Utilizando rectángulos más pequeños

2) Utilizando colores diferentes

1) Para ello modificamos las líneas 50 y 70:

```
50 X2=X1+5
70 Y2=Y1+5
```

Si reejecutamos ahora el programa con RUN, podemos ir contando los 1000 rectángulos que van apareciendo uno tras otro en la pantalla.

Otra posibilidad de modificar el programa consiste en utilizar el subcomando STEP en la instrucción LINE. La línea 80 quedará como sigue:

```
80 LINE(X1,Y1)-STEP(+5,+5),,BF
```

Ahora se pueden borrar las líneas 50 y 70, puesto que X2 e Y2 serán hallados automáticamente en la línea 80 a partir de los valores X1 e Y1.

2) Para conseguir rectángulos de diferentes colores, entre el siguiente programa (o modifique el anterior adecuadamente):

vea listado de LINE 11 del apéndice

Ahora podemos apreciar claramente que el ordenador es capaz de visualizar 1000 rectángulos de cualquier tamaño en la pantalla. A tal fin hemos añadido la línea 80, que escoge aleatoriamente uno de los 16 colores disponibles para cada rectángulo a visualizar.

En este caso también se demuestra la eficacia del sistema MSX gracias a los numerosos subcomandos de que dispone: podemos borrar la línea 80 (el comando COLOR) y sustituirla por el comando LINE ampliado con otro parámetro:

```
90 LINE(X1,Y1)-(X2,Y2),INT(RND(1)*15),BF
```

Pronto se dará cuenta de que se vuelven a producir mezclas de colores en algunas posiciones de la pantalla. Como ya mencioné en otro capítulo, esta circunstancia lamentable no tiene remedio en SCREEN 2, puesto que el modo gráfico de alta resolución está formado por 256 * 192 puntos, pero la resolución del color 'sólo' por 32 * 192. Si varios colores coinciden muy cerca, el último es el color dominante, que suele extenderse sobre varios puntos de pantalla. Sin embargo, con un poco de reflexión y cuidado podemos confeccionar gran variedad de imágenes de hasta 16 colores en el modo gráfico de alta resolución SCREEN 2, disponiendo para ello de 16 Kilobytes de la RAM de vídeo. Otros ordenadores, que ofrecen la misma cantidad de memoria para la creación de gráficos, la representación en alta resolución sólo puede tener 4 ó incluso 2 colores; en estos casos no existe el peligro de las mezclas de colores.

Para finalizar este capítulo, volvemos a experimentar con los colores en dos programas:

1) Muestra la forma en que no debe hacerse, puesto que los diversos colores están demasiado cerca entre sí, provocando una mezcla en la pantalla:

vea listado de LINE 12 del apéndice

2) Esta distancia es correcta para evitar la mezcla de colores, lo que no significa que los diferentes colores deban separarse tanto en cada caso:

vea listado de LINE 13 del apéndice

Si desea tener más información sobre las diferencias entre colores y gráficos de alta resolución en los ordenadores MSX, puede consultar el capítulo que hace referencia al comando SCREEN, donde se explica detalladamente la configuración de la pantalla en el modo gráfico de alta resolución.

LINE 1

```
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(150,150)
40 LINE-(160,120)
50 GOTO 50
```

LINE 2

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 0,0"
40 LINE(128,96)-(150,150)
50 LINE STEP(10,10)-(180,180)
60 LINE STEP(-20,-20)-STEP(-40,-40)
70 GOTO 70
```

LINE 3

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=10 TO 240
40 LINE(128,96)-(N,0)
50 NEXT N
60 GOTO 60
```

LINE 4

```
10 COLOR 1,15
20 SCREEN 2
30 M=241
40 FOR N=10 TO 240 STEP M
```

```
50 LINE(128,96)-(N,0)
60 NEXT N
70 M=M-10
80 IF M<1 THEN GOTO 110
90 CLS
100 GOTO 40
110 GOTO 110
```

LINE 5

```
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(148,96)
40 LINE-(148,76)
50 LINE-(128,76)
60 LINE-(128,96)
70 GOTO 70
```

LINE 6

```
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(148,76)
40 GOTO 40
```

LINE 7

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 1000
40 X1=INT(RND(1)*255)
50 X2=INT(RND(1)*255)
60 Y1=INT(RND(1)*191)
70 Y2=INT(RND(1)*191)
80 LINE(X1,Y1)-(X2,Y2),,B
90 NEXT N
100 GOTO 100
```

LINE 8

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96"
40 FOR N=1 TO 10
50 LINE STEP(0,0)-STEP(10,10),,B
60 NEXT N
70 GOTO 70
```

LINE 9

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96"
40 FOR N=1 TO 10
50 LINE-(128+(N*10),96+(N*10)),,B
60 NEXT N
70 GOTO 70
```

LINE 10

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 1000
40 X1=INT(RND(1)*255)
50 X2=INT(RND(1)*255)
60 Y1=INT(RND(1)*191)
70 Y2=INT(RND(1)*191)
80 LINE(X1,Y1)-(X2,Y2),,BF
90 NEXT N
100 GOTO 100
```

LINE 11

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 1000
40 X1=INT(RND(1)*255)
50 X2=INT(RND(1)*255)
60 Y1=INT(RND(1)*191)
70 Y2=INT(RND(1)*191)
80 COLOR INT(RND(1)*15)
90 LINE(X1,Y1)-(X2,Y2),,BF
100 NEXT N
110 GOTO 110
```

LINE 12

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=100 TO 5 STEP-1
40 LINE(128,96)-STEP(+N,+N),INT(RND(1)*1
5),BF
50 NEXT N
60 GOTO 60
```

LINE 13

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=100 TO 5 STEP-8
40 LINE(128,96)-STEP(+N,+N),INT(RND(1)*1
5),BF
50 NEXT N
60 GOTO 60
```

Comandos Gráficos CIRCLE

Traza círculos, sectores circulares y elipses

vea también los programas del apéndice: CIRCLE 1
CIRCLE 2
CIRCLE 3
CIRCLE 4
CIRCLE 5
CIRCLE 6
CIRCLE 7
CIRCLE 8
CIRCLE 9
CIRCLE 10
CIRCLE 11

CIRCLE es uno de los comandos más potentes del BASIC MSX, que no obstante, dicho de antemano para tranquilizarlo, ya ha llevado a la desesperación a más de un usuario del sistema PAL de televisión, puesto que los círculos dibujados no son tan circulares como deberían ser.

vea listado de CIRCLE 1 del apéndice

Este fallo puede corregirse ya de entrada: sustituya la línea 30 del programa anterior por la siguiente:

```
30 CIRCLE(128,96),50,,,,4/3
```

Al ejecutar ahora el programa con RUN ... observará que el círculo es realmente circular (a menos que por ignorar este factor corrector haya desajustado su televisor en sentido vertical de tal manera que el círculo redondo parezca ahora una ciruela; en este caso, la cabeza del locutor de TV también tendrá forma de ciruela; deberá ajustar nuevamente su televisor).

¿Cuándo podemos utilizar el comando CIRCLE? Cuando nos hallamos en una de las pantallas gráficas (SCREEN 2 ó SCREEN 3); de lo contrario se produce el mensaje de error 'Illegal function call'. ¿Qué significan ahora tantos parámetros?

CIRCLE(A,B),C,D,E,F,G

A y B definen el centro de la circunferencia. Podemos situarlo en el lugar deseado dentro de la matriz de 256*192 puntos (entrada absoluta), o bien a cierta distancia del último punto indicado (con ayuda del subcomando STEP = entrada relativa). Ambas posibilidades se muestran en ejemplos:

1) entrada absoluta:

vea listado de CIRCLE 2 del apéndice

2) entrada relativa:

vea listado de CRICLE 3 del apéndice

Mientras que la entrada absoluta exige la indicación exacta de las coordenadas del centro del círculo, el direccionamiento relativo parte del último punto indicado (que es el 128,96 en CIRCLE 3), considerado como nuevo origen (la entrada CIRCLE comenzará a contar a partir de aquí: 0,0), con ayuda del parámetro STEP.

C indica el radio del círculo a dibujar, que no está limitado al margen de la pantalla (el radio puede tener cualquier valor admitido en la utilización de números enteros (entre +32767 y -32768). Si el valor indicado excede estos límites, se produce un 'Overflow error' en el ordenador MSX).

D indica el código del color utilizado para dibujar el círculo. Debe actuar con precaución al utilizar color en los ordenadores MSX, para evitar feas mezclas de superficies coloridas (puesto que la resolución gráfica se compone de

256*192 puntos, pero la del color tan sólo de 32*192). El siguiente programa le muestra claramente el efecto mencionado:

vea listado de CIRCLE 4 del apéndice

La manera más fácil de evitar estas mezclas de colores consiste en aumentar la distancia entre los diversos círculos. Cambie por ejemplo la línea 30:

```
30 FOR N=1 TO 200 STEP 8
```

E y F son parámetros que definen el principio y el final del círculo respectivamente (de esta forma pueden confeccionarse fácilmente diagramas de sectores, por ejemplo), pero hay que saber utilizarlos correctamente. El problema se debe a la elaboración del ordenador MSX: no admite entradas en grados sino en radianes.

Además hay que acostumbrarse -se quiera o no- a que el BASIC MSX no comienza el trazado del círculo en la parte superior (0 grados), sino en la posición de las tres horas (90 grados), siguiendo el movimiento en el sentido contrario al del reloj. Otra advertencia, antes de aprender el uso del radián en pequeños ejemplos: las indicaciones de principio y final del círculo deben tener valores comprendidos entre +2 PI (+6.21318) y -2 PI (-6.21318); cualquier otro valor es registrado con el mensaje de error 'Illegal function call'.

En primer lugar trazamos medio círculo (180 grados) comenzando en la posición de las tres horas (180 grados equivalen a 1 PI radianes = 3.14159):

vea listado de CIRCLE 5 del apéndice

A continuación dibujamos un cuarto de circunferencia comenzando en la posición de las nueve horas (180 grados):

vea listado de CIRCLE 6 del apéndice

¿Cuál sería el aspecto de un círculo trazado a rayitas? Veamos el ejemplo siguiente:

vea listado de CIRCLE 7 del apéndice

Si ya ha adquirido algunos conocimientos sobre el comando DRAW o el llamado Macro-Graphic-Language (MGL), entenderá ahora la forma de dibujar un sencillo diagrama de sectores dividido en cuatro partes:

vea listado de CIRCLE 8 del apéndice

Antes de confiarle el truco para convertir fácilmente radianes en grados (y también en sentido inverso), quiero enseñarle la representación gráfica de una forma biológica desarrollada por la naturaleza a través de millones de años (compare la superficie interior blanca con la forma de un caracol), y que puede ser imitada en pocos segundos por el ordenador MSX:

vea listado de CIRCLE 9 del apéndice

Bonito, ¿verdad?

Vayamos ahora al método de conversión. ¿Para qué, si no, sirve nuestro potente ordenador MSX? ¡También para efectuar cálculos!

Simplemente dividimos el número de grados por el valor 57.2958 (este valor surge de la correspondencia entre 360 grados y 2 * PI: 360/6.38218 = 57.2958):

180 grados equivalen a PI ó 3.14159
60 grados equivalen a 1.047197 etc.

Podemos definirlo en nuestro programa asignando una función:

```
10 DEF FNA(GRAD)=GRAD/57.2958
20 INPUT "indicar grados ";GRAD
30 PRINT FNA(GRAD)
```

Vayamos ahora al último parámetro del comando CRICLE, que ya hemos conocido y apreciado al principio del presente capítulo:

G define la deformación del círculo hasta trazar una elipse. Puede tomar valores comprendidos entre 1/260 y 260, aunque no tiene interés llegar hasta el extremo de dibujar una línea. Por esta razón se suelen utilizar deformaciones entre 1 (estado normal) y 0 (el círculo se deforma en el sentido del eje X), o bien entre 1 y 2 (el círculo se deforma en el sentido del eje Y).

Para finalizar el capítulo sobre el comando CIRCLE visualizaremos este efecto con ayuda del programa siguiente:

<u>vea listado de CIRCLE 10 del apéndice</u>

Ultimas advertencias: los programas de ejemplo habrán demostrado que no siempre es preciso indicar todos los parámetros del comando CIRCLE. Los únicos datos imprescindibles son: el centro de la circunferencia (A,B) y el radio (C). Para su trazado se acepta el último color utilizado, y los parámetros no indicados expresamente se señalan mediante comas.

Puesto que usted ya no tendrá dificultades para convertir grados en radianes, también entenderá la asignación de variable

A=57.2958

del siguiente programa (línea 50), sin necesidad de volver a explicar el origen de este número:

El programa CIRCLE 11 trabaja con una serie de comandos BASIC, que calculan valores sucesivos para confeccionar un círculo con PSET tras su indicaciones de radio, coordenadas X-Y e incluso distancia entre puntos. También he incluido en el programa un comando CIRCLE de radio inferior y color diferente, para mostrar claramente la rapidez del trazado (en lenguaje máquina) con el comando CIRCLE comparado con la lentitud del programa en BASIC:

vea listado de CIRCLE 11 del apéndice

Sin embargo, como contrapartida, al menos en un aspecto el comando CIRCLE efectuado en BASIC es más completo que el comando CIRCLE en lenguaje máquina: se puede entrar un parámetro adicional, que controla la distancia entre puntos del trazado.

CIRCLE 1

```
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),50
40 GOTO 40
```

CIRCLE 2

```
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),20
40 CIRCLE(64,80),20
50 GOTO 50
```

CIRCLE 3

```
10 COLOR 1,15
20 SCREEN 2
30 PSET(128,96)
40 CIRCLE STEP(-20,0),30
50 CIRCLE STEP(+60,30),30
60 GOTO 60
```

CIRCLE 4

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 100
40 CIRCLE(128,96),N,INT(RND(1)*15)
50 NEXT N
60 GOTO 60
```

CIRCLE 5

```
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),30,,0,3.14159
40 GOTO 40
```

CIRCLE 6

```
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),30,,3.14159,4.712385#
40 GOTO 40
```

CIRCLE 7

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=0 TO 6.21318 STEP 1.55795
40 CIRCLE(128,96),30,,N,N+1
50 NEXT N
60 GOTO 60
```

CIRCLE 8

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=0 TO 4.712385# STEP 1.55995
40 CIRCLE(128,96),30,,N,N+1.55795
50 Z=Z+1
```

```
60 IF Z=1 THEN DRAW "N u30"
70 IF Z=2 THEN DRAW "N l30"
80 IF Z=3 THEN DRAW "N d30"
90 IF Z=4 THEN DRAW "N r30"
100 FOR M=1 TO 100:NEXT M
110 NEXT N
120 GOTO 120
```

CIRCLE 9

```
10 COLOR 1,15
20 SCREEN 2
30 Z=1
40 FOR N=0 TO 6.28318 STEP .1
50 Z=Z+1
60 CIRCLE(128,96),Z,,0,N
70 NEXT N
80 GOTO 80
```

CIRCLE 10

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=2 TO 1/260 STEP -.1
40 CIRCLE(128,96),60,,,,N
50 NEXT N
60 GOTO 60
70 NEXT N
80 GOTO 80
```

CIRCLE 11

```
10 COLOR 15,4
20 CLS
30 INPUT "Radio, X, Y, Distancia entre p
untos  (R,X,Y,P) ";R,X,Y,P
40 SCREEN2
50 A=57.2958
60 CIRCLE(X,Y),R-10
70 FOR B=1 TO 360 STEP P
80 X1=R*COS(B/A)+X
90 Y1=R*SIN(B/A)+Y
100 PSET(X1,Y1),1
110 NEXT B
120 GOTO120
```

Comandos Gráficos PAINT

Colorea áreas limitadas en las pantallas gráficas

vea también los programas del apéndice: PAINT 1
PAINT 2
PAINT 3
PAINT 4

A diferencia de otras variantes del lenguaje BASIC, el MSX no sólo dispone de comandos gráficos para marcar puntos (PSET) o trazar líneas (LINE), sino que además puede representar áreas limitadas en la pantalla (LINE ,,B) y colorearlas con PAINT.

Observemos con qué facilidad pueden visualizarse diversas áreas en la pantalla utilizando los comandos gráficos del MSX:

vea listado de PAINT 1 del apéndice

Este pequeño programa nos muestra cuatro formas diferentes de crear figuras gráficas cerradas: un círculo en el interior (CIRCLE - línea 30); después un rectángulo (LINE ,,B) trazado igualmente con un único comando (línea 40); a continuación otro rectángulo mayor (DRAW - dibujado con ayuda de nuestro lenguaje de programación de lujo en la línea 50); y finalmente otro rectángulo todavía mayor, visualizado en la pantalla por varias líneas unidas entre sí formando ángulos rectos (LINE - líneas 60 a 90).

Sin embargo, no queremos limitarnos a realizar construcciones reticuladas para explicar el comando PAINT; las utilizamos simplemente para colorearlas después. A tal fin aplicamos el comando PAINT, buscando en primer lugar un punto en el interior (!) de la figura. Para colorear el círculo no hace falta ser un genio matemático, ya que su centro (de coordenadas 128,96 en este programa) se encuentra en el interior del mismo. De momento no conocemos todavía la forma de aplicar varios colores con el comando PAINT, por lo que nos limitamos al color utilizado para dibujar la figura.

De forma parecida procedemos coordenada por coordenada y área por área, hasta reunir los siguientes comandos para colorearlas:

1) PAINT(128,96) para el círculo
2) PAINT(85,105) para el rectángulo LINE-B
3) PAINT(75,115) para el rectángulo DRAW
4) PAINT(65,125) para el rectángulo LINE dividido en 4 partes

vea listado de PAINT 1a del apéndice

Al igual que otros comandos gráficos del MSX (LINE, PSET), PAINT tampoco nos obliga a indicar las coordenadas absolutas (por ejemplo, 128,96) del punto donde debe comenzar a colorear el área. También podemos pedir que el mismo ordenador calcule dicho punto de forma relativa. Sin embargo, en nuestro ejemplo eso sólo merece la pena cuando se trata de colorear el círculo de la línea 30: en este caso, el último punto gráfico mencionado es su centro (= 128,96). Si desea comenzar a colorearlo en ese punto, debe modificar la línea 35:

```
35 PAINT STEP (0,0)
```

lo que significa: desplázate desde el centro del círculo (el actual punto del cursor gráfico en ese momento) 0 puntos hacia la derecha y 0 puntos hacia abajo -con lo que se vuelve lógicamente sobre el mismo centro (128,96)- y comienza a colorear la superficie con el color actual.

Tal como mencionamos anteriormente, el comando PAINT suele colorear el área del mismo color que el utilizado para trazar sus límites. Para apreciarlo claramente, cambie el color en cualquier lugar delante (!) del comando PAINT con COLOR. Toda la pantalla se llenará, por ejemplo, de

```
33 COLOR 3
```

y nuestro anterior dibujo ha sido repintado o borrado. Ello

se debe a que el ordenador MSX continúa coloreando hasta hallar un límite del mismo color. Puesto que hasta ahora no habíamos realizado ningún dibujo del color 3, el ordenador colorea toda la pantalla.

Por otra parte es posible utilizar diversos colores para colorear un área, cambiando adecuadamente el del límite. A tal fin entre el siguiente programa en su ordenador:

vea listado de PAINT 2 del apéndice

Se dibujan uno tras otro círculos de diferentes colores, utilizando los mismos también para colorearlos. Sin embargo, vuelven a producirse (¡otra vez!) superposiciones de colores. Por lo tanto debe actuar con precaución al trabajar con tantos colores diferentes a distancias tan reducidas en SCREEN 2, puesto que los ordenadores MSX no admiten más de dos colores distintos en cada patrón de 8*1 puntos.

El programa anterior puede prescindir completamente del comando COLOR de la línea 30, si comunicamos la información referida al color tanto al comando CIRCLE como al PAINT; a tal fin vea el capítulo CIRCLE y el texto siguiente.

Haciendo referencia específicamente al comando PAINT: al igual que otros muchos comandos del MSX, PAINT también dispone de varios parámetros, el primero de los cuales indica el color que debe utilizarse para su función. Modificamos el programa anterior de la siguiente forma:

vea listado de PAINT 2a del apéndice

Este programa produce los mismos efectos que el anterior, con la diferencia de que hemos prescindido de la instrucción COLOR (¡puede ser conveniente en algunos casos, puesto que COLOR hace referencia además a las pantallas de texto SCREEN 0 y SCREEN 1!). Sin embargo, también podemos trabajar indicando específicamente un color de relleno (parámetro 1), y un color para marcar el límite de la figura a colorear (parámetro 2; sin embargo, este último sólo funciona en

SCREEN 3):

PAINT X,Y, color del área, color del límite

Para demostarlo dibujamos tres círculos en la pantalla y, a continuación, coloreamos el círculo inferior utilizando el color de borde del exterior:

vea listado de PAINT 3 del apéndice

Por favor, no se moleste por las superposiciones de colores (como una mezcla de ellos), debidas a las diferentes resoluciones de color y gráfica del ordenador MSX al trabajar en SCREEN 2.

Otras advertencias:

1) Disponemos de 2 comandos para colorear áreas en las pantallas gráficas: PAINT y LINE(...)-(...),,BF. Mientras que el comando LINE-BOX-FILL sólo rellena áreas rectangulares, el comando PAINT puede colorear cualquier superficie. Para conferir color a superficies más grandes, es conveniente utilizar el comando LINE-BF, que convence por su mayor velocidad:

1)
```
10 SCREEN 2
20 PAINT(128,96),1
30 GOTO 30
```
15 segundos

2)
```
10 SCREEN 2
20 LINE (0,0)-(255,191),1,BF
30 GOTO 30
```
1 segundo

2) El comando PAINT comprueba sólo en cuatro direcciones (arriba, abajo, izquierda y derecha) la existencia de un borde. Por esta razón puede atravesar fácilmente los límites dispuestos en diagonal, como ya hemos demostrado, destrozando de esta forma la imagen creada.

El siguiente programa en BASIC muestra la forma en que trabaja el comando PAINT, sin profundizar en cada uno de los comandos gráficos:

vea listado de PAINT 4 del apéndice

Este programa comprueba si ya ha alcanzado el color deseado del borde. Sin embargo, su ejecución es más lenta comparada con el comando PAINT y, a su vez, mucho más lenta que el comando LINE-BF.

Otros dos consejos para los fanáticos de los comandos PEEK-POKE de la memoria: la memoria de trabajo del comando PAINT comienza en la posición $HF949. El color, aceptado por PAINT como el del borde, se memoriza en &HFCB2.

PAINT 1

```
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),10
40 LINE(80,110)-(150,70),,B
50 DRAW " B M 70,120 R90 U60 L90 D60"
60 LINE(60,130)-(170,130)
70 LINE -(170,50)
80 LINE -(60,50)
90 LINE -(60,130)
100 GOTO 100
```

PAINT 1a

```
35 PAINT(128,96)
45 PAINT(85,105)
55 PAINT(75,115)
95 PAINT(65,125)
```

PAINT 2

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 14
40 COLOR N
50 CIRCLE(128,96),50
60 PAINT STEP(0,0)
70 NEXT N
80 GOTO 80
```

PAINT 2a

```
40 REM
50 CIRCLE(128,96),50,N
60 PAINT STEP(0,0),N
```

PAINT 3

```
10 COLOR 1,15
20 SCREEN 3
30 FOR N=1 TO 3
40 CIRCLE(128,96),N*10,N
50 NEXT N
60 PAINT(128,96),3,1
70 GOTO 70
```

PAINT 4

```
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(158,46),,B
40 FOR N=129 TO 157
50 FOR M=95 TO 47 STEP-1
60 IF POINT(N,M)<>1 THEN PSET(N,M)
70 NEXT M
80 NEXT N
90 GOTO 90
```

Comandos Gráficos SPRITE$(N)

Variable dimensionada que contiene los patrones de los sprites

vea también los programas del apéndice: SPRITE$(N) 1
SPRITE$(N) 2
SPRITE$(N) 3
SPRITE$(N) 4
programa 1

La mayor parte de los comandos gráficos sólo pueden aplicarse en las pantallas gráficas SCREEN 2 y SCREEN 3. Al intentar utilizarlos en SCREEN 0 ó SCREEN 1, se emite el mensaje de error 'Illegal function call'. Sin embargo, esta característica varía al trabajar con SPRITE$(N), aunque éste también sea un comando gráfico: no se produce ningún error al utilizarlo en SCREEN 1 (ya sabemos que SCREEN 1 permite representar sprites en la pantalla). Tampoco se emite ningún mensaje de error en SCREEN 0, aunque en este caso no puedan visualizarse los patrones de sprites confeccionados. Entonces, ¿para qué sirve el comando SPRITE$ en SCREEN 0? Este comando proporciona una asignación de variable dimensionada, pero sin ocupar memoria en la RAM normal, sino en la VRAM. Esto significa que al trabajar exclusivamente en SCREEN 0, SPRITE$ le sirve para colocar hasta 32 caracteres en strings, sin necesidad de reservar memoria previamente con CLEAR o de dimensionar una variable con DIM. El segundo parámetro de SCREEN sirve para indicar la cantidad de variables SPRITE$ a utilizar:

SCREEN ,0 y SCREEN ,1:
256 variables de un máximo de 8 caracteres
SCREEN ,2 y SCREEN ,3:
64 variables de un máximo de 32 caracteres

Naturalmente también puede utilizar esta aplicación algo sorprendente del comando SPRITE$(N) (en realidad, se trata más bien de una asignación de variable) en SCREEN 1 a 3, pero en este caso ya no dispondrá de la capacidad completa de definición de sprites (sin embargo, si trabaja sólo con 8

sprites, puede disponer de la memoria VRAM, que corresponde a los sprites restantes, para almacenar variables; en cualquier caso, los valores arriba indicados deben servir de guía).

Vayamos ahora a explicar el significado del propio comando SPRITE$(N): sirve para definir el aspecto o patrón del sprite número N. En un sprite de 8*8=64 puntos aislados, sólo 8 caracteres se encargan de su patrón, por lo que es un poco difícil definir un sprite efectuando las entradas en modo directo (directamente a través del teclado). ¿Cómo se representa exactamente una línea de sprite (8 puntos de pantalla activados o desactivados) a tavés de un carácter? Para ello debe saber que el ordenador concibe cada carácter en forma de un número entre 0 y 255. Puede consultar los capítulos de CHR$ y ASC para obtener más información sobre el tema. Aquí sólo recordamos, que para el ordenador la letra 'A' equivale al número 65.

Ahora debemos retroceder otro paso para descifrar ocho puntos de pantalla activados o desactivados tras el número '65'. A tal fin analizamos la representación de dicho número en letra de ordenador: el MSX convierte los valores del sistema decimal al sistema binario. De esta forma, el resultado de 65 será el número binario 1000001. Pruébelo usted mismo utilizando el comando BIN$ del BASIC MSX:

```
PRINT BIN$(65)
```

Sin embargo, para la representación de 8 puntos, el número binario debe disponer de 8 ceros para los puntos desactivados y de 8 unos para los activados, por lo que deben completarse las primeras posiciones con ceros:

```
PRINT STRING$(8-LEN(BIN$(65)),"0")+BIN$(65)
```

Ahora podrá observar la configuración del primer número de nuestro sprite, en el caso en que la asignación de SPRITE$ comience por 'A'. Esto significa, que en una matriz de sprites de 8*8 precisamos de sólo 8 caracteres, por ejemplo

8 veces la letra 'A', para definir un patrón de 8*8=64 puntos o estados de encendido/apagado.

Sin embargo, la forma habitual de definir sprites no se realiza según el 'trial and error system' (método de tanteos) que acabamos de conocer. Normalmente se diseña un sprite previamente sobre papel y a continuación se determina el respectivo carácter (por ejemplo, una 'A'). A tal fin debe leer la expresión binaria (con 0 y 1) y convertirla después en un carácter. Observemos el siguiente programa:

vea listado de SPRITE$(N) 1 del apéndice

¿Qué proceso ha seguido la definición del sprite?

1) Las líneas 60 a 90 representan el sprite en sus 64 puntos aislados (o sea, ceros para los puntos desactivados y unos para los activados), es decir, allí donde se ha escrito un 1 se encontrará después un punto activado en el sprite, mientras que la posición correspondiente a un 0 quedará vacía.

2) Una tras otra (líneas 30 a 70) se van leyendo las 8 líneas del sprite (línea 40), completado con el código binario (línea 50), y finalmente convertido en carácter y sumado uno por uno al sprite en cuestión (línea 60).

De esta forma, que a primera vista parece complicada, se origina nuestro sprite, que en su configuración final de caracteres ya no se parece a su aspecto inicial:

```
SPRITE$(1)="AUAUAUAU"
```

Ahora ya conoce la forma de definir un sprite ahorrando memoria. Observemos el resultado en la pantalla para comprobar nuestra definición del sprite, ampliando el programa escrito anteriormente:

vea listado de SPRITE$(N) 1a del apéndice

La velocidad con que el ordenador MSX convierte los 8 caracteres en la variedad de puntos visualizada en la pantalla, denota su facilidad y rapidez para efectuar cálculos con ceros y unos.

Hemos mencionado anteriormente, que la representación del tamaño 8*8 permite almacenar hasta 256 patrones de sprites en la VRAM,que podrán ser visualizados en la pantalla con ayuda del comando PUTSPRITE y el número del sprite correspondiente.

Recordará de las explicaciones sobre el comando SCREEN, que su segundo parámetro sirve para predeterminar el tamaño de los sprites:

SCREEN,0

También es posible, acceder a

SCREEN,1

para definir los sprites de 8*8. En este caso aparecerá una versión de 16*16 de sprites, pero cada punto será representado como 2*2 puntos.

Analicemos la diferencia entre SCREEN,1 y SCREEN,2, ampliando nuevamente el programa anterior:

vea listado de SPRITE$(N) 1b del apéndice

También es posible representar sprites de 16*16, compuestos no simplemente por 8*8 puntos ampliados, sino realmente por 16*16 puntos independientes. Para ello debemos acceder a

SCREEN,2 o bien
SCREEN,3

(el último corresponde a la versión ampliada de 16*16). Al definir sprites de este tamaño hay que proceder de manera diferente: a este fin debe saberse que un sprite de 16*16

está formado en realidad por 4 sprites con patrones del tamaño 8*8. Esto significa, que al definir sprites de estas dimensiones, éstos ya no están formados por 8 caracteres, sino por 4*8 = 32, que se juntan de la forma siguiente:

En primer lugar se lee el sprite de 8*8 que se encuentra en la parte superior izquierda, a continuación el sprite 8*8 inferior izquierdo, después el superior derecho y finalmente el sprite inferior derecho.

De esta forma, el proceso de lectura se complica un poco:

vea listado de SPRITE$(N) 2 del apéndice

Si no queremos limitarnos a contemplar la representación de caracteres efectuada después de entrar todo el programa, vayamos a analizar más detenidamente el proceso seguido para definir sprites:

Debemos ampliar el programa con SCREEN,2 y PUTSPRITE para conseguir una representación normal de 16*16.

Si además deseamos ampliar la versión del sprite de tamaño 16*16, debemos entrar el comando SCREEN,3 al principio del programa.

También es posible visualizar nuestro sprite gigante en una matriz de sólo 8*8 puntos en SCREEN,0 ó SCREEN,1. A tal fin entre al principio del programa

```
SCREEN ,1
```

o bien,

```
SCREEN ,2
```

para ampliar la representación de 8*8.

Sin embargo, los dos últimos modos utilizados sólo pueden representar 8*8 puntos, por lo que veremos sólo la esquina superior izquierda del sprite. Es decir, el sprite definido de 16*16 puntos se reduce automáticamente al patrón de 8*8 del sprite superior izquierdo.

Observemos el proceso seguido en la lectura de un sprite de 16*16: en primer lugar, una variable de cadena dimensionada previamente lee toda la información sobre los patrones (línea 40). A continuación, todas las primeras mitades de los sprites serán interpretadas y empleadas en la definición del sprite. Después se efectúa el mismo proceso con las 8 cifras de definición del sprite de la derecha (contando de forma decimal). De esta forma se origina finalmente nuestro sprite de 16*16 (compuesto en realidad por 4 sprites de matriz de 8*8 puntos).

La VRAM puede memorizar hasta 64 sprites grandes de 16*16 (comparado con los 256 del tamaño de 8*8), aunque no puedan visualizarse más de 32 sprites al mismo tiempo en la pantalla. La elección es suya.

Otras indicaciones interesantes:

1) Podemos leer en la VRAM el lugar donde se han almacenado patrones de sprites: a tal fin nos servimos de las variables de indicación BASE(8) para SCREEN 1, BASE(13) para SCREEN 2 y BASE(18) para SCREEN 3, sumándoles 8 para llamar cada uno de los patrones. Por ejemplo, para visualizar las 8 cifras de las líneas o expresiones binarias del sprite 1 en SCREEN 1, entre el siguiente programa:

vea listado de SPRITE$(N) 3 del apéndice

Para conseguir la visualización de un sprite de 16*16 leyendo directamente en la VRAM, la cosa se complica. Observemos más detenidamente el proceso seguido por la conversión binaria:

vea listado de SPRITE$(N) del apéndice

2) Vaya con precaución, si desea cambiar el tamaño de los sprites en medio de un programa. Cualquier modificación de SCREEN efectuado con el fin de cambiar de SCREEN (acceder a su primer parámetro) o el tamaño de los sprites (acceder al segundo parámetro) borra todos los patrones de sprites de la VRAM. También existe otra forma de hacerlo, pero ésta exige profundizar mucho más en la estructura del ordenador MSX.

vea programa 1 del apéndice

3) En cualquier momento puede consultar la cantidad de memoria de sprites (indicación en cantidad de caracteres) que queda disponible:

```
PRINT SPRITE$( )
```

Si trabaja con la matriz de sprites de 16*16, deberá dividir la cantidad indicada por 8 para obtener el número de patrones no definidos todavía. Si utiliza, en cambio, las versiones de 16*16 puntos, el valor emitido por PRINT SPRITE() debe dividirse por 16 para obtener el mismo resultado.

SPRITE$(N) 1

```
10 COLOR 1,15
20 SCREEN 1
30 FOR N=1 TO 8
40 READ A$
50 A$="&B"+A$
60 B$=B$+CHR$(VAL(A$))
70 NEXT N
80 SPRITE$(1)=B$:PRINT SPRITE$(1)
90  DATA 01000001
100 DATA 01010101
110 DATA 01000001
120 DATA 01010101
130 DATA 01000001
140 DATA 01010101
150 DATA 01000001
160 DATA 01010101
```

SPRITE$(N) 1a

```
85 PUT SPRITE 1,(128,96)
```

SPRITE$(N) 1b

```
15 IF Z=1 THEN Z=0 ELSE Z=1
20 SCREEN 1,Z
25 RESTORE
86 LOCATE 0,0
87 PRINT "Tamano de sprite";Z
88 FOR O=1 TO 1000
89 NEXT O:GOTO 15
```

SPRITE$(N) 2

```
10 COLOR 1,15
20 SCREEN 1
30 DIM A$(16)
40 FOR N=1 TO 16
50 READ A$(N)
60 NEXT N
70 FOR N=1 TO 16
80 A$=A$+CHR$(VAL("&B"+LEFT$(A$(N),8)))
90 NEXT N
100 FOR N=1 TO 16
110 A$=A$+CHR$(VAL("&B"+RIGHT$(A$(N),8))
)
120 NEXT N
130 SPRITE$(1)=A$
140 DATA 0101010101010101
150 DATA 1010101010101010
160 DATA 0101010101010101
170 DATA 1010101010101010
180 DATA 0101010101010101
```

```
190 DATA 1010101010101010
200 DATA 0101010101010101
210 DATA 1010101010101010
220 DATA 0101010101010101
230 DATA 1010101010101010
240 DATA 0101010101010101
250 DATA 1010101010101010
260 DATA 0101010101010101
270 DATA 1010101010101010
280 DATA 0101010101010101
290 DATA 1010101010101010
```

SPRITE$(N) 3

```
10 FOR N=14336+32 TO 14336+39
20 PRINT STRING$(8-LEN(BIN$(VPEEK(N))),"
0")+BIN$(VPEEK(N))
30 NEXT N
```

SPRITE$(N) 4

```
10 FOR N=14336+32 TO 14336+47
20 PRINT STRING$(8-LEN(BIN$(VPEEK(N))),"
0")+BIN$(VPEEK(N));
30 PRINT STRING$(8-LEN(BIN$(VPEEK(N+16))
),"0")+BIN$(VPEEK(N+16))
40 NEXT N
```

Modifica los atributos (color,posición,etc) de los sprites

vea también los programas del apéndice: PUTSPRITE 1
MISTER MINER
FRETY
programa 4
programa 5

El ordenador MSX ofrece la posibilidad de almacenar hasta 256 sprites de patrones diferentes en la memoria RAM de vídeo, y de visualizar en la pantalla hasta 32 sprites al mismo tiempo.

Para definir los sprites nos servimos de la asignación de variable mediante el comando SPRITE$, mientras que la llamada del mismo hacia un punto determinado de la pantalla se efectúa con el comando PUTSPRITE, que significa tanto como: coloca el sprite en una posición determinada. Finalmente, con ayuda de un control de interrupción podemos comprobar si dos sprites han colisionado en la pantalla (ON SPRITE GOSUB).

Los sprites pueden representarse en cualquier otra pantalla, además de SCREEN 0. Sin embargo, hay que tener en cuenta el tamaño de sprites que deseamos escoger a través del segundo parámetro del comando SCREEN. Si son los grados 0 y 1 (SCREEN ,0 ó bien SCREEN ,1), podemos definir hasta 256 sprites del tamaño de un campo de 8*8 (tamaño de letra). Si trabajamos con los grados 2 y 3 (SCREEN ,2 ó bien SCREEN ,3), tan sólo podemos confeccionar 64 sprites del tamaño 16*16.

Analicemos detalladamente el formato del comando PUT SPRITE:

```
PUT SPRITE A,(B,C),D,E
```

Con respecto a A: Sirve para escoger un valor entre 0 y 31, comunicando así al ordenador el plano en que deseamos trabajar. Debe imaginarse los 32 diferentes sprites como

anversos transparentes que, colocados en la pantalla, simplemente muestran sus patrones definidos en el lugar deseado. El sprite 31 se encuentra directamente en nuestra imagen creada, mientras que el sprite 30 ya se halla en un plano superior. Esto significa que, cuando dos sprites se van moviendo y finalmente se superponen, el sprite 31 se oculta debajo del sprite 30. De la misma forma se dispone toda la jerarquía de posibles sprites 0 a 31. El número inferior siempre tiene mayor prioridad, por lo cual se encuentra por encima del sprite con número superior. Volviendo a la formulación: por cada plano de la pantalla, o sea por cada anverso transparente, puede representarse sólo 1 sprite en cada caso; es decir, que debemos visualizar todos los planos de sprites -de 0 a 31- al mismo tiempo si queremos aprovechar la capacidad de representación de figuras móviles.

vea programa 4 del apéndice

Con respecto a B y C: Sirven para definir las coordenadas X e Y respectivamente del sprite a visualizar. No estamos limitados a los valores dentro del campo visual de la pantalla. B y C también pueden tomar valores entre -32768 y +32767 (valores que exceden de estos límites producen un 'Overflow error'); el sprite en cuestión volverá a aparecer continuamente en la pantalla en determinados intervalos. Por un lado podemos utilizar el direccionamiento absoluto, indicando por ejemplo los valores B=128 y C=96 para acceder al centro de la pantalla:

```
PUT SPRITE 1,(128,96)
```

Pero también se admite (al igual que en todos los comandos gráficos) un direccionamiento relativo. A tal fin consideramos el actual cursor gráfico como origen e indicamos al ordenador la cantidad de puntos que deberá moverse hacia la derecha/izquierda y hacia abajo/arriba. Nuestra preferencia por el método relativo debe transmitirse al ordenador a través de la palabra STEP (=paso) integrada en el comando PUT SPRITE. Por ejemplo, para desplazar el

cursor 10 puntos hacia la derecha y 10 hacia arriba, la instrucción será:

```
PUT SPRITE A,STEP(10,-10),D,E
```

Por cierto: en cualquier caso (sprites grandes o pequeños), las coordenadas indicadas en el direccionamiento tanto absoluto como relativo, hacen referencia a la esquina superior izquierda extrema del patrón del sprite (¡no al primer punto visible del patrón que usted haya confeccionado!).

Con respecto a D: Cada sprite debe ser de un único color. El parámetro D sirve para indicar el código del color deseado, comprendido entre 0 y 15.

Los sprites de varios colores sólo pueden conseguirse superponiendo varios sprites de diferentes colores. De esta forma podría crearse, por ejemplo, un muñequito rojo (sprite 0) con chaqueta verde (sprite 1) y pantalones azules (sprite 2):

vea programas FRETY y MISTER MINER del apéndice

Sin embargo, debe tener precaución al trabajar con sprites de varios colores:

a) El comando ON SPRITE ya no podrá aplicarse, puesto que los sprites de varios colores se superponen continuamente y envían de forma ininterrumpida el mensaje sobre la colisión de figuras al VDP del ordenador.

b) En el BASIC, el control de varios sprites de colores superpuestos puede ser tan lento que incluso permita apreciar, por ejemplo, que en primer lugar se desplaza la figura roja, después la chaqueta y finalmente los pantalones. Es bastante difícil evitar este efecto.

c) Hay que actuar con precaución al componer un muñequito de tres o más sprites diferentes, puesto que el ordenador MSX

sólo admite cuatro sprites por línea. Al añadir un quinto sprite, el de grado superior será borrado o tapado, cosa que puede producir efectos poco atractivos.

vea programa 5 del apéndice

Se preguntará por qué ya existe tal peligro con muñequitos formados por sólo 3 sprites. Piense que en los juegos a veces se encuentran dos muñequitos en la misma línea de pantalla ...

A pesar de ello, en los juegos también pueden conseguirse figuras móviles de varios colores. Los dos programas de juego incluidos en este libro trabajan de esta forma. Quizás pueda aprender algo cuando vaya a teclearlos.

vea programas FRETY y MISTER MINER del apéndice

Con respecto a E: El segundo parámetro del comando SCREEN indica al ordenador el tamaño con que debe visualizar los sprites en la pantalla: puede elegir entre definir sprites de 8*8 (SCREEN ,0 ó bien SCREEN ,1) o sprites de 16*16 (SCREEN ,2 ó bien SCREEN ,3) y emitirlos en la pantalla.

Puesto que la zona de memoria destinada a los sprites es limitada, el ordenador MSX sólo admite hasta 256 diferentes sprites de 8*8, o bien 64 sprites de 16*16. En la pantalla pueden visualizarse simultáneamente 32 de estos sprites mediante PUT SPRITE.

El parámetro E del comando PUT SPRITE sirve para indicar cuál de las figuras definidas previamente con SPRITE$ deberá aparecer en la pantalla. Puede decidir si todos los 32 sprites corresponden a un mismo patrón o a 32 patrones diferentes de la memoria. Para ello pruebe el siguiente programa:

vea listado de PUTSPRITE 1 del apéndice

También en este caso debe tener precaución: si se visualizan 5 sprites en la misma horizontal, el de mayor valor será borrado parcial o totalmente (¡en sentido vertical no existe límite!).

vea programa 5 del apéndice

Otras advertencias: la existencia de más de 4 sprites en una horizontal puede ser controlada a través de la lectura de VDP(7). El valor emitido por un bit de VDP(7) (vea capítulo de VDP) hace referencia al número del sprite de mayor valor, que aparecerá entonces en la pantalla borrado parcial o totalmente (habrá desaparecido).

En lo referente a los patrones de sprites del comando PUT SPRITE, advertimos que también puede entrar comas para saltarse la indicación del color:

```
PUT SPRITE 1,(128,96),,1
```

que significa: se visualizará el patrón del sprite 1 almacenado en la memoria. En este caso, la figura móvil tomará el color del primer plano válido en ese momento.

En los casos de omisión, siempre se utilizan los últimos valores indicados, tanto para el color como para el número del patrón del sprite.

Si conoce las coordenadas actuales de su sprite (B,C), podrá modificar solamente su color, sin desplazarlo en la pantalla:

```
DRAW "BM =B;,=C;"
PUT SPRITE 1,STEP(0,0),color
```

El comando BASE le informa sobre el lugar donde comienzan dos posiciones de memoria VRAM vitales para los sprites:

1) La tabla de atributos de sprites, que contiene los datos referentes al número del patrón, coordenadas actuales y

color del sprite (estos son los datos que reclamamos con PUT SPRITE a través de BASE).

2) La tabla de patrones de sprites, que contiene información sobre el aspecto de los sprites (datos creados con SPRITE$ a través de BASE).

Si prefiere no utilizar los comandos PUTSPRITE y SPRITE$ del BASIC, también puede colocar los valores deseados mediante VPOKE en las direcciones correspondientes de dichas tablas, o bien leer en la memoria la información necesaria con VPEEK (por ejemplo, ¿coordenadas y color del sprite 3 en ese momento?). Sin embargo, este acceso directo a la memoria exige un conocimiento perfecto sobre la configuración de la VRAM; con respecto a ello recomendamos consultar los capítulos BASE, SCREEN y VDP de este libro.

Se puede eliminar a corto plazo cualquier sprite de la pantalla, indicando el valor 209 como ordenada Y:

```
PUT SPRITE 1,(X,209)
```

Efectuando cualquier entrada legal como ordenada Y, el sprite en cuestión volverá a aparecer en el lugar correspondiente de la pantalla.

Los sprites de número superior pueden desaparecer a corto plazo de la pantalla indicando el valor 208 en lugar de la ordenada Y:

```
PUT SPRITE 1,(X,208)
```

que significa: todos los sprites de número superior a 1 desaparecerán instantáneamente de la pantalla.

Puede anular este efecto indicando, en nuestro caso para el sprite 1, un valor legal como ordenada Y.

PUTSPRITE 1

```
10 COLOR 1,15
20 SCREEN 1,0
30 VDP(6)=VDP(4)
40 PRINT "Primero 5 sprites con"
50 PRINT "patrones diferentes:
60 FOR N=0 TO 191
70 PUT SPRITE 1,(128,N+8)
80 PUT SPRITE 2,(128,N+16)
90 PUT SPRITE 3,(128,N+24)
100 PUT SPRITE 4,(128,N+32)
110 PUT SPRITE 5,(128,N+40)
120 NEXT N
130 CLS
140 PRINT "Ahora 5 sprites con"
150 PRINT "el mismo patron:"
160 FOR N=0 TO 191
170 PUT SPRITE 1,(128,N+8),,1
180 PUT SPRITE 2,(128,N+16),,1
190 PUT SPRITE 3,(128,N+24),,1
200 PUT SPRITE 4,(128,N+32),,1
210 PUT SPRITE 5,(128,N+40),,1
220 NEXT N
230 RUN
```

Comandos Sonido PLAY

Comando de control para la programación musical del MSX

vea también los programas del apéndice: programa [illegible]
programa 9
programa 10
PLAY(N) 4

Además del gran número de comandos de sonido y, sobre todo, gráficos, el BASIC del MSX ofrece también dos lenguajes propios programables, que facilitan considerablemente el acceso a cualquier efecto con los procesadores gráfico y de sonido:

Para gráficos, el GML (Graphic Macro Language)
Para sonido, el MML (Music Macro Language)

Mientras que el lenguaje gráfico trabaja con el comando DRAW, comentado en otro capítulo, el lenguaje musical se sirve del comando PLAY, que quiere decir algo así como:'toca notas'.

Ambos lenguajes, gráfico y musical, trabajan con strings, lo que significa que los subcomandos deben estar cerrados entre comillas detrás de PLAY y DRAW. Sin embargo, esta propiedad permite asignar los sonidos definidos a diversas variables, las cuales podrán ser transformadas a conveniencia mediante comandos de string (MID$, LEFT$, INSTR, RIGHT$ ...) y ampliadas hasta 255 caracteres por string.

No es necesario separar los diversos subcomandos contenidos en el string de PLAY mediante coma o punto y coma. Sólo en algunos casos, que indicaremos debidamente, será preciso utilizar tal separación. Sin embargo hemos dejado espacios entre los subcomandos explicados en este capítulo,para facilitar la lectura de las instrucciones. Puede eliminarlos escribiendo los caracteres directamente uno tras otro. Esto es especialmente interesante en los programas largos,puesto que permite ahorrar memoria.

Pruebe crear sonido con su ordenador MSX. Al entrar la instrucción

PLAY "CDE"

sonarán las notas C,D y E. De esta forma puede escribir hasta 255 notas en un string.

Si desea que suenen dos secuencias de notas paralelamente, deberá separarlas mediante coma en forma de dos strings independientes:

PLAY "CDE","DEF"

y sonarán paralelamente las notas CD, DE y EF. En este caso también sería posible teóricamente hacer sonar 255 sonidos uno tras otro y paralelamente entre sí; a través de un comando PLAY.

El límite de las posibilidades del MSX consiste en tocar tres notas al mismo tiempo. En este caso, el tercer canal también se separará de los dos primeros mediante coma:

PLAY "C","E","G"

Al igual que indicamos antes, se pueden tocar paralelamente hasta 255 notas por canal musical mediante un único comando PLAY.

Al utilizar PLAY disponemos de las siguientes designaciones de sonidos (vea también el manual de referencia del MSX):

C D E F G A B

Las teclas negras de un piano (los semitonos entre teclas blancas contiguas) corresponden a los signos positivo (+) y negativo (-) en el Music Macro Language.

Una escala musical completa sin semitonos tendría esta forma:

PLAY "CDEFGAB"

y aquí una escala con todos los semitonos:

PLAY "CC+DD+EE+FF+GG+AA+BB+"

Se habrá dado cuenta de que aquí falla algo, y tiene toda la razón:

1) El tono "E+" corresponde al tono F
2) El tono "B+" corresponde al tono C

La imagen de una octava del teclado de un piano nos lo muestra claramente:

vea programa 9 del apéndice

Una octava consta exactamente de siete tonos enteros y cinco semitonos. La "E" elevada no existe, sino que en su lugar se encuentra la nota "F" siguiente, contigua a la "E" y, por lo tanto, de igual valor.

Si queremos tocar un semitono superior a la "B", tampoco encontramos ninguna tecla negra correspondiente; el siguiente tono posible es "C", pero éste ya corresponde a una octava superior a la que estamos tratando.

Para corregir la escala anterior, a continuación se muestra la octava correcta junto a otra posibilidad de escribir los semitonos:

1) PLAY "CC+DD+EFF+GG+AA+B"
2) PLAY "CD-DE-EFG-GA-AB-B"

Como es habitual en la notación musical, en lugar del signo positivo (para señalar un semitono elevado) también podemos indicar una doble cruz, con lo que la anterior secuencia de

tonos tomará el siguiente aspecto:

1) PLAY ''CC≠DD≠EFF≠GG≠AA≠B''

Para indicar un paso de semitono siempre hay que colocar un signo positivo, negativo o una doble cruz -tal como ha podido ver en los ejemplos anteriores- detrás de la nota en cuestión; en caso contrario, dicho signo haría referencia a la nota inmediatamente precedente (si existe).

Hace un momento hemos descubierto nuestro error anterior con ayuda del teclado de un piano en el programa 8: sabemos que no todos los semitonos añaden un sonido nuevo a los tonos existentes. ¿Y qué sucede con nuestra escala anterior, donde "B+" volvió a emitir el sonido de un "C" bajo?

Debemos indicar al ordenador la octava en que se halla el sonido que queremos emitir. Nuestro ejemplo anterior no contiene ninguna indicación al respecto, de forma que el ordenador entiende el valor estándar, es decir: toca en la octava 4. Pero en realidad podemos disponer de 8 octavas al mismo tiempo, que señalaremos añadiendo una "O" y un número entre 1 y 8 al comando PLAY. Probemos ahora a emitir la "B+" realmente elevada con esta nueva instrucción, o sea, a tocar el tono "C" de la octava superior:

PLAY "CC+DD+EFF+GG+AA+B O5 B+"

¡Estupendo, ya sale ... y no vuelve a salir si se toca de nuevo! Si mueve el cursor hacia la línea escrita anteriormente en la pantalla y vuelve a ejecutarla, sonará la misma secuencia de tonos pero tocados en la octava 5, lo que significa que la última nota desciende de nuevo a la "C" baja. Después de leer el comando "O5", el ordenador sabe que a partir de ese momento debe tocar todas las notas en la octava 5. Podríamos sustituir ahora la octava 5 por la octava 6 ("O5" se convierte en "O6"), pero con ello tampoco solucionamos nada, porque la escala irá elevándose continuamente hasta llegar a la octava 8.

Otra solución más eficaz consiste en colocar el número de la octava 4 delante de las primeras notas tocadas, pudiendo mantenerse la indicación "05" (una octava superior a la "04"):

```
PLAY "04 CC+DD+EFF+GG+AA+B 05 B+"
```

Esta es la instrucción correcta y podrá cambiar de octava como quiera (desde la octava 1, que suena más o menos grave, hasta la octava 8 con los sonidos más elevados). Puede indicar la octava mediante su correspondiente número junto a "O" delante de cada una de las notas, o bien indicarla sólo una vez delante de la adecuada secuencia de notas. El ordenador MSX entiende naturalmente ambas expresiones.

Al principio del capítulo hemos creado una composición de tres voces en conjunto. En cualquier caso, el valor indicado de la octava, al igual que las demás variaciones del sonido mencionadas en este capítulo, sólo es válido para el canal de sonido correspondiente. Si deseamos extender la modificación a todos los canales, tendremos que utilizar los mismos comandos en cada uno de ellos: de esta forma, los 3 canales siguientes emitirían el mismo sonido de tres voces con diferentes alturas de sonido:

```
PLAY "04 C 05 C 06 C","04 E 05 E 06 E","04 G 05 G 06 G"
```

Sin embargo, recuerde que para mantenerse dentro de una misma octava, no debe superar nunca "B" (puesto que "B+" corresponde a la "C" baja, de la misma octava) ni "C" ("C-" corresponde a la "B" alta).

En el Macro Music Language también es posible trabajar con números correspondientes a los diversos sonidos: estos números deben señalarse en el comando PLAY con una "N" (de number = número) precedente. Esto significa que podemos acceder a cualquier nota de todas las octavas disponibles a través de un número, es decir, la "C" de la primera octava es el número 1 ("N1") y la "B" de la octava 8 es el 96 ("N96"). Puesto que cada octava está formada por 12 tonos

("CC+DD+EFF+GG+AA+B") y que podemos disponer de 8 octavas, existen en total 8*12=96 tonos. Como orientación diremos que la "C" de la octava 4 corresponde a "N36", y la octava completa expresada en números sería:

```
PLAY "N36 N37 N38 N39 N40 N41 N42 N43 N44 N45 N46 N47"
```

Seguramente se preguntará ahora para qué necesita esta indicación bastante confusa además de la anterior más clara que expresaba las notas. Las razones son varias:

1) No debemos preocuparnos de indicar la octava adecuada. La entrada se efectúa con valores numéricos absolutos, es decir: a mayor número le corresponde mayor altura de sonido.

2) En el transcurso de este capítulo conoceremos la forma en que el ordenador puede procesar los strings (partirlos en strings parciales de PLAY así como unirlos), para lo cual será más fácil trabajar con valores numéricos que con nombres de notas, cuyo orden es más difícil de entender por el ordenador (puede calcular y entender claramente la secuencia "CDEFG", porque sigue el orden del alfabeto; ¿pero qué ocurre con las notas "A" y "B" siguientes?). No es tan fácil explicarle al ordenador que la G va seguida de una A.

Hasta este momento, el ordenador MSX ha tocado todas las notas de igual duración, al igual que un niño que aprende sus primeras notas de violín. Sin embargo, además de la altura de los sonidos, también se puede variar su duración: después de conectar el ordenador, cada nota suena con un cuarto de duración, que se expresa con el subcomando "L" de PLAY de la siguiente forma:

```
PLAY "L4"
```

Mientras que "L4" designa un cuarto de nota, también podemos poner "L8" delante de una nota, lo que significa que las notas siguientes (hasta encontrar otro cambio de duración mediante "L") son octavos de nota. "L1" significaría notas enteras, "L2" medias notas ... hasta llegar a "L64" = 1/64

de nota (una nota entera se compone en este caso de 64 notas de duración 1/64).

El ejemplo siguiente hace sonar los tres canales de sonidos con la misma duración, pero cada sonido tiene una duración diferente:

canal 1 = 8 veces 1/8 de nota
canal 2 = 4 veces 1/4 de nota
canal 3 = 1 vez 1/1 de nota

Para poder apreciar y así oir la diferencia, asignamos una octava diferente a cada canal:

```
PLAY "04 L8 CDECDECD","05 L4 CDEC","06 L1 C"
```

De forma parecida al comando de octavas, nuestro ordenador MSX también memoriza la duración del sonido hasta volver a cambiarla. Sin embargo, si dichos cambios afectan a notas aisladas, eliminamos la indicación "L" y la sustituimos por la duración deseada del sonido, colocada detrás de la nota en cuestión. De esta forma los sonidos siguientes tendrían igual altura y duración:

```
PLAY "C32"
PLAY "L32 C"
```

El segundo ejemplo indica mediante "L32", que todas las notas posteriores (hasta la siguiente entrada efectuada con "L") deben tener igualmente una duración de 1/32 de nota.

También se puede prescindir de la letra "L" al efectuar cambios de notas aisladas. En este caso, sin embargo, debemos colocar un punto y coma para separar los diversos valores numéricos (tanto por la altura del sonido, como por su duración):

Entrando notas: PLAY "C8 DE"
Entrando números: PLAY "N36;8 N38 N40"

Además existe otra forma de variar la duración de un sonido: al igual que en la notación musical, un punto añadido a una nota determina que la duración de la misma aumenta en su mitad. De esta forma, la longitud de una nota entera se convierte en una longitud y media, la de media nota en 3/4 de longitud, etc:

PLAY "L4 C." = tres cuartos de nota
PLAY "L1 C." = una nota entera y una mitad

Para hacer música necesitamos también la posibilidad de no hacerla. Estas interrupciones (pausas, en inglés "rest") se señalan con la letra "R" en las secuencias de sonidos. Le sigue por su parte un número entre 1 (pausa entera) y 64 (1/64 de pausa):

PLAY "C R4 DE"

Suena la nota "C", después hay una pausa de 1/4 y finalmente escucharemos los sonidos "D" y "E".

Si deseamos tocar varias veces el mismo tono uno tras otro, es conveniente entrar valores de pausas extremadamente cortas: mientras que los 4 tonos del ejemplo siguiente suenan como un único tono largo:

PLAY "CCCC"

la corta pausa ("R64") provoca, que pueda distinguirse cada uno de los 4 tonos:

PLAY "C R64 C R64 C R64 C"

Sin embargo, si utiliza este método al entrar música de varias voces, pronto se resentiría de la emisión asincrónica de los tres canales de sonido.

También podemos utilizar la letra "R" en el string sin indicar ningún número. De esta forma "R" tomará su valor estándar, que corresponde a una pausa que dura 1/4 de nota.

Antes de proseguir con la variación directa del sonido (curva envolvente, etc.), debemos explicar el significado de tiempo y volumen. Hasta ahora hemos hablado de medias notas, de cuartos y 1/64 de nota y hemos considerado que un cuarto de nota debe sonar las mismas fracciones de segundo en cualquier pieza musical. Sin embargo, esto no es correcto, puesto que el tiempo (movimiento rítmico de la batuta del director) también determina la velocidad con que debe tocarse cierta pieza con sus notas.

Sin embargo, dentro de un tiempo determinado, las notas tienen la misma duración relativa:

> una 1/2 nota tiene igual longitud que dos de 1/4 de nota
> una nota entera tiene igual longitud que ocho de 1/8 etc

Al conectar el ordenador MSX, el tiempo tiene valor 120. Este valor puede reclamarse en cualquier momento, entrando simplemente la letra "T" sin más indicación numérica. Por otro lado, también es posible incrementar el tiempo de nuestra pieza (hasta "T255") o bien reducirlo (hasta "T32"). Observemos, o mejor dicho, escuchemos este efecto con ayuda del siguiente ejemplo:

```
PLAY "T CDE"         tiempo = 120 = normal
PLAY "T240 CDE"      tiempo = 240 = rápido
PLAY "T60 CDE"       tiempo = 60 = lento
```

El volumen de la música puede graduarse mediante la letra "V" (del inglés "volume" = volumen) y un número entre 0 y 15. Si no realiza ningún comando "V", o si utiliza "V" sin indicación numérica en un string musical, los sonidos siguientes sonarán con volumen=8, mientras no se indique lo contrario.

El volumen sube con "V" hasta 15 y baja con "V" hasta 1, o bien "V" hasta 0, que corresponde a silencio:

```
PLAY "V CDE"         volumen = 8 = normal
PLAY "V15 CDE"       volumen = 15 = alto
PLAY "V1 CDE"        volumen = 1 = bajo
PLAY "V0 CDE"        volumen = 0 = apagado
```

Tal como se explica en el capítulo de SOUND, podemos variar el timbre de cada sonido con ayuda de diferentes curvas envolventes. Por ejemplo, es posible hacer que un sonido no suene inmediatamente, sino que su volumen aumente lentamente (hasta llegar a su valor máximo). También se puede hacer que el sonido se eleve y vuelva a descender.

Para efectuar esta variación del sonido, disponemos de 8 diferentes curvas envolventes a través de la letra "S" (del inglés "Shape"). Para hacer sonar una curva envolvente, es aconsejable graduar la longitud de sonido a 1 (L1=nota entera), saltarse la indicación de volumen (V) (puesto que el ordenador MSX trabaja continuamente con V16 al dirigirse a las curvas envolventes) y añadir algunos sonidos (para poder escuchar alguna cosa, porque la curva envolvente suena únicamente con, o bien, a través del sonido). Se podría efectuar, por ejemplo, la siguiente entrada:

```
PLAY "S10 L1 CDEFG"
```

Escuche detenidamente las siguientes curvas envolventes:

S1, S4, S8, S10, S11, S12, S13, S14

Los tonos "C", "D", y "E" convertidos en un ruido de helicóptero mediante una curva envolvente se conseguiría, por ejemplo, de la forma siguiente:

```
PLAY "S8 CDE"
```

o el ruido de tambores:

```
PLAY "S1 CDE"
```

Además puede afinar cada curva envolvente, efectuando una

sintonización determinada con ayuda de la letra "M" seguida de un número entre 1 y 65535.

De esta forma, los tambores del ejemplo anterior aumentan su volumen y timbre:

PLAY "S1 M2000 CDE"

En el capítulo de SOUND se explicarán las razones de este efecto. De momento sólo queremos mencionarlo.

Hasta aquí hemos citado todas las posibilidades de utilizar convenientemente el comando PLAY, o mejor dicho 'el lenguaje de programación PLAY'. A continuación se muestra un resumen de todos los parámetros:

C D E F G A B	nombre de sonido
+ - ≠	variación de altura por un semitono
O	elección entre 8 octavas
N	número de altura de sonido (1 a 96)
L	duración de sonido (1=1/1 a 64=1/64)
.	aumentar duración de sonido en su mitad
R	pausa (1=1/1 a 64=1/64)
T	tiempo (32 a 255)
V	volumen (0 a 15)
S	curva envolvente (1,4,8,10,11,12,13,14)
M	sintonizar curvas envolventes (1 a 65535)

Seguramente habrá notado, que su ordenador MSX es capaz de tocar una secuencia de sonidos (también a tres voces en conjunto), mientras que el cursor ya está preparado para la siguiente entrada. Ello se debe a que el ordenador MSX posee una memoria separada, que puede memorizar hasta 24 sonidos (a tres voces) y tocarlos sucesivamente sin perturbar la ejecución del programa. Sin embargo, si supera este límite de memoria, puede tardar bastante tiempo hasta que la ejecución del programa vuelva a su velocidad normal (vea también PLAY(N)).

vea listado de PLAY(N) 4 del apéndice

Además recordará que el comando PLAY trabaja con strings normales. De esta forma se pueden tocar uno tras otro varios strings definidos previamente, tal como muestra el siguiente ejemplo:

```
10 A$="CDE"
20 B$="FGA"
30 PLAY A$,B$
```

Otra posibilidad consiste en introducir dichos strings directamente en instrucciones PLAY existentes:

```
PLAY "CDE X B$;"
```

Para ello, el string debe ir precedido de una "X" y seguido de un punto y coma (";").

El comando PLAY también puede leer variables directamente del programa (de forma parecida a la inserción de un subcomando, la variable va precedida de un "=" y seguida de un ";"), por ejemplo, a continuación sonarán todos los 96 tonos de "N1" hasta "N96":

```
10 FOR N=1 TO 96
20 PLAY "N =N;"
30 NEXT N
```

La velocidad de ejecución del programa aumenta si tiempo y duración del sonido toman sus valores máximos:

```
20 PLAY "T255 L64 N =N;"
```

En la línea 20, la variable leída "N" (del bucle FOR ... NEXT de la línea 10) está cerrada entre un signo de igualdad y un punto y coma, que sirven para separarla del resto del string.

Otra advertencia: La combinación de teclas µCTRL° µSTOP°, para interrumpir un programa, así como el comando BEEP

devuelven todos los parámetros de PLAY (octava, volumen, ...) a su estado original (octava = 4, volumen = 8, ...). Es interesante saberlo para cuando no sepa cómo salirse de las variaciones de PLAY-SOUND que haya efectuado.

Además, cabe señalar que tanto el comando SOUND como BEEP y PLAY acceden a los mismos registros del procesador de sonido. Esto significa que, aparte de algunas excepciones, la entrada de uno de dichos comandos influye también sobre las variaciones de salida de otro (por ejemplo, BEEP reactiva todos los registros de sonido a su valor inicial).

Comandos Sonido PLAY(N)

Comprueba si se está tocando música

vea también los programas del apéndice: PLAY(N) 1
PLAY(N) 2
PLAY(N) 3
PLAY(N) 4

Al hablar del lenguaje de programación PLAY ya mencionamos, que los ordenadores MSX poseen una zona separada de la memoria, donde puede memorizar hasta 25 sonidos (a tres voces). Cuando el lenguaje PLAY haya explicado estos sonidos al ordenador, la ejecución del programa en BASIC continúa, mientras que en el fondo suena la música leída.

Este proceso se efectúa de forma natural para nosostros. Sin embargo, no deberíamos olvidar que el BASIC MSX nos ofrece algo muy particular, con esta capacidad de ejecutar dos procesos al mismo tiempo. En realidad, el procedimiento seguido es el llamado control de interrupciones, que quiere decir: El BASIC es interrumpido continuamente por una rutina en lenguaje máquina, sin que ello afecte a nuestros programas en BASIC.

El BASIC MSX se distingue por un gran número de controles de interrupciones, que continuamente comprueban por una parte una posible colisión entre sprites (ON SPRITE), por otra la pulsación de una tecla de función (ON KEY), y finalmente la reacción adecuada del programa cuando aparezca un error (ON ERROR).

Volvamos ahora al comando PLAY. El control de interrupciones y el de los 25 sonidos son controlables, es decir, no sólo nuestro oído puede apreciar el final de la melodía tocada (o sea, cuando se terminan los 25 sonidos que suenan en el fondo), sino que además podemos comprobarlo en la memoria. Para ello sirve el comando PLAY(N) o, hablando correctamente, la variable PLAY(N): si se toca algún sonido a través de cualquiera de los canales de sonido disponibles, la respuesta es afirmativa, que en el lenguaje del ordenador

significa = -1; en caso contrario, la respuesta es negativa = 0. ¡Probémoslo!

vea listado de PLAY(N) 1 del apéndice

Mientras la música continuaba sonando se mostró el número -1 en la pantalla (-1 = verdadero = suena música). Cuando finalmente paró de sonar la música, la pantalla mostró el valor 0 (0 = falso = la música ya no suena).

Al igual que en otros muchos comandos -o mejor dicho comprobaciones mediante variables- del MSX, en este caso tampoco debemos limitarnos a comprobar mediante la instrucción PLAY(N) simplemente si la música suena o no.

Los diversos parámetros sirven para hacer las siguientes distinciones:

PLAY(0): lee los 3 canales por encendido/apagado, -1/0
PLAY(1): lee canal 1 por encendido/apagado, -1/0
PLAY(2): lee canal 2 por encendido/apagado, -1/0
PLAY(3): lee canal 3 por encendido/apagado, -1/0

En una pieza musical puede ocurrir fácilmente que el canal 1 continúe tocando la melodía principal, mientras que los canales 2 y 3 estén preparados para recibir nuevas instrucciones musicales. Este procedimiento se muestra en la pantalla mediante el programa siguiente:

vea listado de PLAY(N) 2 del apéndice

En la línea 40 suena una melodía de tres voces, donde cada canal presenta un número diferente de sonidos a tocar, con el fin de demostrar los efectos mencionados. Al ejecutar el programa con RUN, en primer lugar se muestra la palabra "encendido" cuatro veces en la pantalla; los canales dejan de tocar uno tras otro, lo que muestra la correspondiente reacción en la pantalla ("apagado").

Quizás le sorprenda la comprobación IF ... THEN de las líneas 90 a 120, por lo que indicamos que:

IF A THEN...

significa: si se cumple A, o sea si A=-1 (es decir, al menos uno de los tres canales de sonido está activado), escribe "encendido" en la pantalla. Según el álgebra booleana, el ordenador MSX entiende que se cumple o no la condición exigida, reaccionando en consecuencia.

La comprobación mediante PLAY aquí explicada,nos permite llenar el canal de sonido hasta tal punto, que el programa pueda continuar su ejecución,sin ser interrumpido durante mucho tiempo por la música que se está tocando.

Comprobemos si los tres canales trabajan realmente independientes durante la emisión de 25 tonos:

vea listado de PLAY(N) 3 del apéndice

Finalizaremos con otra advertencia, que puede servirle de desafío:

En la posición 63830 de la RAM comienza la zona que almacena los datos del comando PLAY. Intente averiguar lo que sucede en esta zona, mientras hace sonar una melodía a través de PLAY.

Un consejo para decodificar parte de esta zona de la memoria.

Puede interrumpir los canales 1, 2 y 3 mientras están tocando música, efectuando los siguientes pasos:

1) Apagar el canal 1:

POKE 63833,255 POKE 63834,255

2) Apagar el canal 2:

POKE 63839,255 POKE 63840,255

3) Apagar el canal 3:

POKE 63845,255 POKE 63846,255

Puesto que estas posiciones de memoria tienen valor 0 en el momento en que no suene ninguna nota, puede sustituir el comando PLAY(N) por una lectura de dichas posiciones. Simplemente active un bucle de comprobación y además del comando PLAY(N), indicará exactamente la cantidad de sonidos que han sido tocados en cada canal hasta ese momento, así como la cantidad que será tocada.

vea listado de PLAY(N) 4 del apéndice

PLAY(N) 1

```
10 COLOR 1,15
20 SCREEN 0
30 PLAY "cdefgab"
40 PRINT PLAY(0)
50 GOTO 40
```

PLAY(N) 2

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH30
40 PLAY "cdefgab","defgab","efgab"
50 A=PLAY(0)
60 B=PLAY(1)
70 C=PLAY(2)
80 D=PLAY(3)
90 IF A THEN PRINT "sonoro ";ELSE PRINT
"mudo   ";
100 IF B THEN PRINT "sonoro ";ELSE PRINT
 "mudo   ";
110 IF C THEN PRINT "sonoro ";ELSE PRINT
 "mudo   ";
120 IF D THEN PRINT "sonoro "ELSE PRINT
"mudo   "
130 GOTO 50
```

PLAY(N) 3

```
10 CLEAR 2000
20 COLOR 1,15
30 SCREEN 0
40 WIDTH37
50 FOR N=1 TO 25
60 A=INT(RND(1)*96)
70 B=INT(RND(1)*96)
80 B=INT(RND(1)*96)
90 A$=A$+"N"+STR$(A)
100 B$=B$+"N"+STR$(B)
110 C$=C$+"N"+STR$(C)
120 NEXT N
130 PLAY A$,B$,C$
140 PRINT "A partir de ahora sonaran por
 si so- los 25 tonos en 3 canales; mient
ras  puede continuar programando (o pued
e ir contando los sonidos)"
```

PLAY(N) 4

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 37
40 PLAY "cdedcdeccdecdecdecdecdec","cede
decdedededededcdede","cdedededededededede"
50 LOCATE 0,0
60 PRINT "Tono numero"INT(PEEK(63834!)/5
)"suena en canal 1"
70 LOCATE 0,1
80 PRINT (PEEK(63833!)-1)/5"Tonos suenan
 en canal 1"
90 PRINT
100 PRINT "Tono numero"INT(PEEK(63840!)/
5)"suena en canal 2"
110 PRINT (PEEK(63839!)-1)/5"Tonos suena
n en canal 2"
120 PRINT
130 PRINT "Tono numero"INT(PEEK(63846!)/
5)"suena en canal 3"
140 PRINT (PEEK(63845!)-1)/5"Tonos suena
n en canal 3"
150 PRINT
160 IF PEEK(63833!)=PEEK(63834!) THEN PR
INT "Canal 1 esta apagado  " ELSE PRINT
"Canal 1 esta encendido"
170 IF PEEK(63839!)=PEEK(63840!) THEN PR
INT "Canal 2 esta apagado  " ELSE PRINT
"Canal 2 esta encendido"
180 IF PEEK(63845!)=PEEK(63846!) THEN PR
INT "Canal 3 esta apagado  " ELSE PRINT
"Canal 3 esta encendido"
190 GOTO 50
```

Comandos Sonido SOUND

Escribe directamente en los registros del PSG (generador de sonido programable)

Existen varias posibilidades de crear sonido con el ordenador MSX: se puede emitir un pitido con el comando BEEP (tal como suena también al producirse un error); se puede conectar y desconectar el chasquido de las teclas con ayuda del tercer parámetro del comando SCREEN; y PLAY permite tocar notas en tres canales al mismo tiempo.

A diferencia de ello, el comando SOUND permite dirigirse directamente al PSG (Programable Sound Generator). Para ello dispone de 14 registros, cuya función analizaremos con más detalle:

Registro 0, posibilidad de entrar p.e. SOUND 0,200

A través de este registro puede determinarse lo grave o agudo que debe sonar un tono en el canal de sonido A. Sin embargo, ello sólo tendrá efecto si previamente se ha ordenado en el registro 7, que deberá emitirse un sonido por el canal A, así como graduado su volumen en el registro 8. Puede escoger diferentes graduaciones en los registros 0 a 255 (de 0 a 255).

Registro 1, posibilidad de entrar p.e. SOUND 1,10

Este registro sirve igualmente para indicar lo grave o agudo que deberá sonar el tono en el canal A, aunque no ofrece una distinción de altura dividida tan finamente como ocurre con el registro 0. Por esta razón, con el registro 1 sólo pueden efectuarse 16 diferentes graduaciones (de 0 a 15) de la altura de sonidos, pero con cada entrada de SOUND efectuada aquí se salta una octava completa.

Los registros 0 y 1 deben considerarse como pareja; el registro 1 sirve para efectuar el ajuste aproximativo (de 0 a 15) y el registro 0 para la sintonización fina de la altura de sonidos.

Al igual que se relaciona la pareja de registros 0 y 1 (ajustes de precisión y aproximativo de la frecuencia del canal A), también sucede con las parejas de registros 2 y 3 así como 4 y 5:

El registro 2 se encarga del ajuste de precisión de la frecuencia del canal B (valores 0 a 255).

El registro 3 se encarga del ajuste aproximativo de la frecuencia del canal B (valores 0 a 159).

El registro 4 se encarga del ajuste de precisión de la frecuencia del canal C (valores 0 a 255).

El registro 5 se encarga del ajuste aproximativo de la frecuencia del canal C (valores 0 a 15).

Como ya hemos mencionado al trabajar con el canal A, un sonido sólo puede crearse si:

1) los canales correspondientes han sido graduados en el registro 7 (registro de mezcla) para la reproducción del sonido (otras posibilidades: zumbido, zumbido más sonido, nada).

2) se ha indicado el volumen del canal A, B y C en los registros 8, 9 y 10, respectivamente, con valores comprendidos entre 1 y 15.

Registro 6, posibilidad de entrar p.e. SOUND 6,5

El comando SOUND no sólo permite crear sonidos, sino también diversos zumbidos. A tal fin, en el registro 7 deben indicarse los canales que deben emitir sonido o zumbido.

Si hemos decidido, por ejemplo, que el canal A efectúe el zumbido, el registro 6 permite conseguir una reproducción de hasta 8 estados diferentes (entrada 0 a 7).

Registro 7, posibilidad de entrar p.e. SOUND &B10101010

Como puede ver, hemos preferido utilizar la notación binaria para representar la entrada en el registro 7. Naturalmente también es posible entrar el valor indicado del ejemplo en forma decimal o hexadecimal; sin embargo, en este caso no se podría reconocer la función de este parámetro de SOUND con suficiente claridad.

A través de la entrada en los tres bits menos significativos (en nuestro ejemplo '010') se indica cuál de los 3 canales está destinado a la emisión del sonido. En el ejemplo anterior, el '1' indica que el canal B crea un sonido, mientras que los dos ceros determinan que los canales A y C no emitirán sonido alguno.

Los tres bits siguientes (en el ejemplo '101') indican los canales que crean zumbido. En el ejemplo anterior, por lo tanto, los canales A y C (ambos activados a '1'=activado) emiten un zumbido, mientras que el canal B emite un sonido.

Los dos bits más significativos siempre deben estar activados a '10'; por lo demás, esta secuencia no tiene mayor importancia.

Analicemos otro ejemplo para explicar el significado del séptimo parámetro:

SOUND 7,&B10100110

En este caso sólo puede emitirse sonido a través de los anales B y C (secuencia '110'), mientras que el canal C crea además un zumbido (secuencia '100'). Esto significa que, con ayuda del séptimo parámetro de SOUND, podemos emitir sonido y zumbido simultáneamente a través de un único canal de sonido.

Registro 8, posibilidad de entrar p.e. SOUND 8,30

Este registro sirve para definir el volumen del sonido o

zumbido emitido por el canal A. Podemos entrar valores entre 0 (apagado) y 15, subiendo el volumen con valor creciente. Si indica valores superiores a 15, el volumen se mantiene al máximo posible, pero puede reproducir además sonidos extraños al mismo tiempo, definidos en los registros 11 a 13 del comando SOUND. Una curva envolvente sólo puede reproducirse si indicamos el volumen correspondiente (en el caso del registro 8, esta entrada se refiere al canal A).

Los registros 9 y 10 trabajan de la misma forma que el 8, pero las entradas efectuadas en ellos hacen referencia a los canales B y C:

El registro 9 se encarga del volumen (0 a 15) o la activación de la curva envolvente (16) de canal B.

El registro 10 se encarga del volumen (0 a 15) o la activación de la curva envolvente (16) de canal C.

Es decir, con las entradas siguientes podemos conseguir crear sonidos en el canal A, zumbido en el canal B, y en el C una curva envolvente con un sonido definido previamente:

```
SOUND 1,10
SOUND 5,12
SOUND 6,4
SOUND 7,&B10010101
SOUND 8,10
SOUND 9,10
SOUND 10,10
```

Registro 11, posibilidad de entrar p.e. SOUND 11,11

Los registros 11 y 12 trabajan conjuntamente de forma parecida a los registros 0 y 1. Mientras que el registro 11 sirve para efectuar el ajuste de precisión de la curva envolvente, el 12 se encarga de su ajuste aproximativo. Ambos registros sólo pueden ejercer su función si previamente hemos definido una forma determinada de curva envolvente en el registro 13, y activado el volumen a 16 con

los registros 8, 9 ó bien 10, para activar así la emisión de la curva.
En este caso, los registros 11 y 12 provocan que la frecuencia de la curva aumente al incrementar los valores de las entradas.

Ambos registros, 11 y 12, admiten entradas entre 0 y 255, por lo que disponemos de 65535 posibilidades de ajuste.

Registro 13, posibilidad de entrar p.e. SOUND 13,4

Existen muchas posibilidades de representar un sonido de diversas formas: un sonido puede tener volumen constante desde su inicio hasta su extinción (como una bocina de coche), o puede elevarse y descender de forma ondulatoria (como una sirena) o ... Con ayuda del chip de sonido del MSX, el PSG (generador de sonido programable), podemos crear 8 curvas envolventes distintas:

0: El sonido es fuerte al iniciarlo y se va extinguiendo lentamente

4: El sonido va creciendo lentamente y desparece repentinamente al llegar a la amplitud de la onda

8: El sonido es fuerte al principio, baja y después vuelve a subir, etc.

10: El sonido baja, después sube, vuelve a bajar, etc.

11: El sonido baja de volumen y suena fuerte después de una corta pausa (sin crecer)

12: El sonido crece lentamente, después casi se apaga, a continuación vuelve a crecer muy lentamente, etc.

13: El sonido crece lentamente y se mantiene finalmente a volumen máximo.

14: El sonido crece lentamente, después decrece de nuevo,

crece otra vez (parecido a la curva envolvente 10, pero aquella tiene su máximo volumen al principio).

Resumamos ahora las funciones de los diversos parámetros de SOUND asOí como los límites de los valores a entrar:

Registro 0: ajuste de precisión de la frecuencia del canal A (0 a 255)
Registro 1: ajuste aproximativo de la frecuencia del canal A (0 a 15)
Registro 2: ajuste de precisión de la frecuencia del canal B (0 a 255)
Registro 3: ajuste aproximativo de la frecuencia del canal B (0 a 15)
Registro 4: ajuste de precisión de la frecuencia del canal C (0 a 255)
Registro 5: ajuste aproximativo de la frecuencia del canal C (0 a 15)
Registro 6: Diversos tipos de zumbido (0 a 7)
Registro 7: los 3 bits menos significativos: activado/desactivado de tono/canal
los próximos 3 bits: activado/desactivado de zumbido/canal
los 2 bits más significativos: siempre '10'
Registro 8: volumen canal A (0 a 15) ó 16 = activación de la curva envolvente
Registro 9: volumen canal B (0 a 15) ó 16 = activación de la curva envolvente
Registro 10: volumen canal C (0 a 15) ó 16 = activación de la curva envolvente
Registro 11: ajuste de precisión de la frecuencia de curva envolvente (0 a 255)
Registro 12: ajuste aproximativo de la frecuencia de curva envolvente (0 a 255)
Registro 13: elección curva envolvente 0,4,8,10,11,12,13,14

No basta con activar la curva envolvente, por ejemplo, sólo con el registro 13. Debemos proceder de forma diferente en cada caso diferente.

Caso 1: Debe emitirse un sonido a través del canal A:

1) Activar registro 0 para la altura de sonido
2) Posiblemente activar registro 1 para la altura de sonido
3) Activar registro 7 de forma que se emita un sonido a través del canal A
4) Entrar en registro 9 un valor entre 1 y 15 para indicar el volumen

Caso 2: Debe emitirse un zumbido a través del canal B:

1) Graduar registro 6 a un tipo de zumbido
2) Activar registro 7 de forma que pueda emitir zumbido pero no sonido
3) Entrar en registro 9 un valor entre 1 y 15 para indicar el volumen

Caso 3: A través del canal C debe emitirse un sonido de sirena que eleve y descienda el tono

1) Activar registro 4 para la altura de sonido
2) Posiblemente activar registro 5 para la altura de sonido
3) Activar registro 7 de forma que se emita un sonido a través del canal C
4) Entrar 16 en registro 10 para activar la curva envolvente
5) Graduar registros 11 y 12 para la frecuencia de la curva
6) En registro 13, escoger la curva envolvente adecuada (seguramente número 10 ó 14).

La ventaja de definir el sonido con ayuda del comando SOUND es que una vez activado el sonido (también a tres voces) continuará emitiéndose hasta que se ordene su interrupción con el propio ordenador MSX. De esta forma puede imaginarse, por ejemplo, que durante un juego se pueda oir continuamente el susurro de olas o también el ruido de un helicóptero, mientras que la ejecución del programa no se interrumpe.

Por cierto:

A través del registro 7 puede activar el zumbido en

determinados canales. Si después intentase reproducir sonidos mediante PLAY, su ordenador MSX indica que ha memorizado las entradas. Ya no sonará ninguna música sino sólo un zumbido.

Si entra el comando BEEP o interrumpe la ejecución de un programa de forma que se emita un pitido, todos los registros de SOUND definidos hasta ese momento quedarán borrados. Deberá comenzar de nuevo efectuando las entradas.

Si desea obtener resultados más espectaculares, puede calcular la frecuencia del sonido con la siguiente fórmula:

para canal A: 124797/256*registro 1+registro 2
para canal B: 124797/256*registro 3+registro 4
para canal C: 124797/256*registro 5+registro 6

Unicamente puede escribir informaciones en el registro 14 de SOUND, pero no podrá volver a leerlas. Con ayuda del comando PLAY(N) puede efectuarse un control parcial: sustituyendo N por el número del canal (0 a 2), sabemos si en este momento suena música (verdadero=-1) o no (falso=0).

Comandos Sonido BEEP

Genera un pitido de atención

vea también los programas del apéndice: BEEP 1
BEEP 2
BEEP 3

Si utilizamos este comando, se estremece más de un programador, puesto que el sonido que genera corresponde exactamente al provocado por el ordenador cuando tropieza con un error en el programa y emite un mensaje en la pantalla (por ejemplo, 'Syntax error in 10').

Puede experimentarlo entrando simplemente caracteres desordenados (sin número de línea) y pulsando a continuación la tecla ENTER.

Podemos provocar el mismo pitido de atención de las siguientes formas:

1) PRINT CHR$(7)

2) Pulsando las teclas CTRL y G

3) PLAY "L64 M255 06 S1 T120 V8 E"

4) Con ayuda del siguiente programa:

vea listado de BEEP 1 del apéndice

Ahora conocemos seis formas (incluyendo el mensaje de error y la entrada directa del comando BEEP) de provocar un pitido. Realmente demasiadas, pensará usted. Sin embargo, no todos los pitidos que se emiten, aunque suenen igual, tienen el mismo significado. ¡Pruébelo!

vea listado de BEEP 2 del apéndice

Coloque uno tras otro los demás generadores del pitido (PRINT CHR$(7), PLAY y el programa SOUND) en la línea 20 (o

el programa de sonido en las líneas 20 a 25) y compruebe las diferencias.

Seguramente se habrá dado cuenta de lo siguiente: cuando entramos una instrucción BEEP o PRINT CHR$(7),se reactivan los registros de sonido de los comandos PLAY y SOUND (en este caso la octava definida). Sin embargo, si utilizamos el pitido generado artificialmente a través del comando PLAY o del programa SOUND, el ordenador MSX recuerda las graduaciones anteriores, es decir, en este caso se mantienen los valores de los registros tal como los habíamos definido inicialmente. Por esta razón es aconsejable utilizar la instrucción PLAY o el programa SOUND, para crear pitidos dentro de un programa. De esta forma continuamos teniendo todos los registros de sonido directamente bajo control.

El pitido de error que aparece repentina e inesperadamente puede eliminarse, si al principio del programa disponemos un control de interrupción ON ERROR para los mensajes de error previsibles.

Otra advertencia: no sé si ha entendido correctamente mi elección de palabras al hablar de 'pitidos de atención'. El siguiente ejemplo con el comando BEEP sirva para explicarlo:

vea listado de BEEP 3 del apéndice

Si en este programa entra un número inferior a 0.6 superior a 50, sonará el pitido, o sea un sonido, que debe llamarle la atención sobre el error cometido (un 'pitido de atención').

BEEP 1

```
10 SOUND 0,85
20 SOUND 1,0
30 SOUND 7,8
40 SOUND 8,7
50 FOR N=1 TO 10
60 NEXT N
70 SOUND 7,9
```

BEEP 2

```
10 PLAY "O6 CEG"
20 FOR N=1 TO 1000
30 NEXT N
40 BEEP
50 PLAY "CEG"
```

BEEP 3

```
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Escriba un numero"
40 INPUT "comprendido entre 0 y 50 ";A
50 IF A<0 OR A>50 THEN BEEP:GOTO 30
60 PRINT "Bien hecho!"
70 GOTO 30
```

Comandos Gráficos STICK

Lee la dirección del joystick y de las teclas del cursor

vea también los programas del apéndice: STICK 1
STICK 2
STICK 3
FRETY
MISTER MINER

Tal como permite adivinar el nombre de este comando, se trata de la lectura relacionada con un dispositivo de control posiblemente conectado en uno de los dos ports de joysticks del ordenador MSX.

En relación a ello ya conocemos los comandos STRIG (botón de fuego pulsado si/no), PAD (haciendo referencia a un "touch pad") y PDL (referente a un "paddle" conectado).

El comando STICK del MSX permite leer la dirección tomada mediante un joystick. Puesto que no sólo pueden conectarse joysticks al ordenador, el comando STICK debe especificar su uso con ayuda de un parámetro:

```
PRINT STICK(0) = lee las teclas del cursor
PRINT STICK(1) = lee joystick en port 1
PRINT STICK(2) = lee joystick en port 2
```

Con ayuda de estas tres lecturas pueden participar hasta tres jugadores en un juego al mismo tiempo: mientras que dos jugadores mueven su joystick en la dirección adecuada, el tercero debe colocarse ante el teclado y utilizar las teclas del cursor para dirigir sus movimientos.

Recordemos: incluso la lectura del botón de fuego con ayuda de los comandos ON STRIG y STRIG podía hacer referencia simultáneamente, además de los dos joysticks conectados, también a la barra de espacio pulsada o no pulsada. Es decir: el ordenador MSX admite el juego de tres jugadores junto con todas sus acciones posibles de control. Pero eso no es todo: controlando las interrupciones con ayuda de ON

STRIG podrá interrumpir el programa únicamente cuando un jugador haya pulsado el botón de fuego o la barra de espacio.

Antes de mostrar la función del comando STICK con ejemplos prácticos, analicemos el significado de los resultados obtenidos por la lectura. A tal fin observemos las direcciones y sus correspondientes valores:

0 = ninguna dirección (posición de salida = reposo, joystick en posición central)

1 = movimiento arriba a la derecha o dirección NE (en el teclado pulsar simultáneamente las flechas hacia arriba y hacia la derecha)

2 = movimiento hacia la derecha o dirección E

3 = movimiento hacia derecha y abajo o dirección SE (en el teclado pulsar simultáneamente las flechas derecha y abajo)

4 = movimiento hacia abajo o dirección S

5 = movimiento abajo a la izquierda o dirección SO (en el teclado pulsar simultáneamente las flechas hacia abajo y hacia la izquierda)

6 = movimiento hacia la izquierda o dirección O

7 = movimiento hacia arriba a la izquierda o dirección NO (en el teclado pulsar simultáneamente flechas izquierda y arriba)

8 = movimiento ascendente o dirección N

Tal como ya mencionamos en el resumen anterior, los valores intermedios a través del teclado sólo pueden conseguirse pulsando dos teclas de dirección al mismo tiempo: para

moverse hacia arriba a la derecha, debe pulsar simultáneamente las teclas hacia arriba y hacia la derecha del cursor.

Por otro lado, si pulsa inintencionadamente dos teclas de sentidos opuestos, el ordenador MSX lo ignorará. Si pulsa, por ejemplo, las teclas hacia izquierda y hacia derecha al mismo tiempo, el resultado será = 0, y una tercera tecla de cursor pulsada, por ejemplo flecha hacia abajo, será interpretada como si las anteriores no se hubiesen tocado. Por lo tanto, en este caso se mostraría el número '5' referente al jugador a través del teclado.

Analicemos ahora dos ejemplos prácticos, que deben servir para explicar la capacidad del comando STICK:

1) Programa de pintura con ayuda del joystick o del teclado

Pero antes tres advertencias:

1- Este ejemplo demuestra que los números 0 (teclado), 1 (joystick 1) y 2 (joystick 2) pueden sustituirse por variables. De esta forma puede permitir en su programa, que el propio usuario tome la decisión de jugar con joystick o con teclado.

2- Si el movimiento del punto que va dibujando la imagen le parece demasiado lento, puede incrementar simplemente los valores tras el comando PSET STEP (por ejemplo, por dirección NE: PSET STEP (+2,-2).

3- Aunque el ordenador MSX no tenga en cuenta si nos estamos moviendo dentro de la matriz de 256*192 puntos, en cualquier caso es mejor reactivar en los programas todas las coordenadas de los extremos a 0, para conseguir una imagen realista en la pantalla y evitar que el cursor se aleje de la superficie visible de la pantalla.

vea listado de STICK 1 del apéndice

2) Lectura de la dirección del joystick

Aparte de algunas excepciones, actualmente todos los joysticks están en peligro de rotura. Para no desconcertar a su proveedor (algunas veces no funciona nada, pero otras sí) el siguiente programa sirve para investigar con ayuda de un gráfico las direcciones de su joystick que funcionan o no correctamente:

vea listado de STICK 2 del apéndice

Naturalmente, este programa también sirve para comprobar la correcta función de las teclas del cursor de su ordenador MSX. Sin embargo, este método es demasiado complicado, puesto que tal comprobación puede efectuarse directamente (teclas de cursor como editores).

Otra advertencia respecto a la lectura del teclado efectuada con PRINT STICK(0): puesto que cada tecla pulsada corresponde a un código numérico, que puede ser leído con ayuda de PRINT ASC(A$), disponemos de dos posibilidades para leer las teclas del cursor:

1- Lectura a través de PRINT STICK(0) (vea más arriba)

2- Lectura del código de la tecla pulsada, tal como se muestra en el siguiente programa:

vea listado de STICK 3 del apéndice

Las teclas de cursor tienen los códigos siguientes:

derecha = CHR$(28)	izquierda = CHR$(29)
arriba = CHR$(30)	abajo = CHR$(31)

Sin embargo, el último programa presenta un problema. Al consultar las direcciones intermedias, el resultado está formado por dos valores, correspondientes a las respectivas direcciones que componen la intermedia.

STICK 1

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 PRINT " Teclado              = 0
          Port del joystick 1 = 1
          Port del joystick 2 = 2"
50 PRINT
60 INPUT " Efectue se entrada (0 a 2) ";
A
70 IF A<0 OR A>2 THEN RUN
80 SCREEN 2
90 PSET(128,96)
100 B=STICK(A)
110 IF B=1 THEN PSET STEP(0,-1)
120 IF B=2 THEN PSET STEP(+1,-1)
130 IF B=3 THEN PSET STEP(+1,0)
140 IF B=4 THEN PSET STEP(+1,+1)
150 IF B=5 THEN PSET STEP(0,+1)
160 IF B=6 THEN PSET STEP(-1,+1)
170 IF B=7 THEN PSET STEP(-1,0)
180 IF B=8 THEN PSET STEP(-1,-1)
190 GOTO 100
```

STICK 2

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 PRINT "Teclado              = 0
         Port del joystick 1 = 1
         Port del joystick 2 = 2"
50 PRINT
60 INPUT "Efectue su entrada (0 a 2) ";A
70 IF A<0 OR A>2 THEN RUN
80 ON STRIG GOSUB 240,240,240,240,240
90 FOR N=0 TO 4
100 STRIG(N) ON
110 NEXT N
120 SCREEN 2
130 B=STICK(A)
140 IF B=0 THEN GOTO 130 ELSE:PSET (128,
96)
150 IF B=1 THEN DRAW "U16 N F8 G8"
160 IF B=2 THEN DRAW "E16 N D8 L8"
170 IF B=3 THEN DRAW "R16 N H8 G8"
180 IF B=4 THEN DRAW "F16 N U8 L8"
190 IF B=5 THEN DRAW "D16 N H8 E8"
200 IF B=6 THEN DRAW "G16 N U8 R8"
```

```
210 IF B=7 THEN DRAW "L16 N E8 F8"
220 IF B=8 THEN DRAW "H16 N D8 R8"
230 GOTO 130
240 CLS
250 RETURN
```

STICK 3

```
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 PRINT "Compruebe las teclas del curso
r"
50 A$=INKEY$
60 IF A$="" THEN GOTO 50
70 LOCATE 10,10
80 IF ASC(A$)=28 THEN PRINT "derecha  "
90 IF ASC(A$)=29 THEN PRINT "izquierda"
100 IF ASC(A$)=30 THEN PRINT "arriba   "
110 IF ASC(A$)=31 THEN PRINT "abajo    "
120 GOTO 50
```

Comandos Gráficos STRIG

Lee hasta cuatro botones de fuego y la barra espaciadora

vea también los programas del apéndice: STRIG 1
STRIG 2
FRETY
MISTER MINER

Otro comando que obtiene su función, o mejor dicho, sus informaciones a través de los ports de joysticks y/o del teclado. A diferencia de STICK (lectura de direcciones), PAD (lectura del "touch pad") y PDL (lectura del paddle), el comando STRIG no se preocupa en leer las direcciones, sino 'sólo' de determinar el estado de pulsado/no pulsado.

Se comprueba si ha sido pulsado un botón de fuego del joystick o/y la barra espaciadora del teclado. Para controlar dicha comprobación, utilizamos la lectura de interrupción efectuada con ON STRIG GOTO. De esta forma se evita la continua espera a la pulsación de la barra espaciadora o de un botón de fuego mediante bucles largos y penosos.

No basta con entrar simplemente PRINT STRIG, sino que para la lectura debemos determinar exactamente:

PRINT STRIG(0) = ¿ha sido pulsada la barra espaciadora?

PRINT STRIG(1) = ¿ha sido pulsado el botón de fuego 1 del joystick 1?

PRINT STRIG(2) = ¿ha sido pulsado el botón de fuego 1 del joystick 2?

PRINT STRIG(3) = ¿ha sido pulsado el botón de fuego 2 del joystick 1?

PRINT STRIG(4) = ¿ha sido pulsado el botón de fuego 2 del joystick 2?

Habrá que reparar en lo siguiente: es aconsejable que la comprobación sobre la pulsación o no pulsación del botón de fuego 2 se efectúe sólo si poseemos realmente un joystick con dos botones de fuego que correspondan a la norma del MSX. Si confecciona un juego destinado a una posterior venta al público, es conveniente crear un menú inicial que admita la opción de jugar con un joystick normal (con un único botón de fuego); piense que no todos los propietarios de un ordenador MSX poseen además un joystick adaptado al cien por cien, que disponga de dos botones a comprobar independientemente. Si su juego admite incluso tres jugadores, es justo anular la comprobación mediante STRIG del segundo botón de fuego de los joysticks, puesto que el teclado sólo dispone de una opción, la lectura de la barra espaciadora.

A continuación analizaremos a través de un pequeño programa los efectos causados por dos comandos del MSX emparentados entre sí, el ON STRIG GOSUB y el PRINT STRIG:

vea listado de STRIG 1 del apéndice

Se habrá fijado en dos particularidades de este programa:

1) Se utiliza un comando STRIG adicional, que activa y desactiva la lectura de interrupción a efectuar con ON STRIG:

```
STRIG ON o bien
STRIG OFF
```

2) La pulsación de un botón de fuego se comprueba a través de la indicación de verdadero (-1) = pulsado botón de fuego o barra espaciadora, o falso (0) = no pulsado botón de fuego o barra espaciadora. IF STRIG(0) THEN significa: si la primera parte de la pregunta ha sido contestada correctamente, ejecuta también la siguiente instrucción; en caso contrario ve hacia la siguiente línea del programa.

Otra advertencia: no sobrevalore la lectura de la barra

espaciadora (pulsada o no) a través del comando STRIG. El siguiente programa demuestra, que dicha lectura también puede efectuarse a través de la instrucción

```
IF ASC(A$)=32
```

vea listado de STRIG 2 del apéndice

Sin embargo, en este programa ya no puede utilizarse el comando ON STRIG GOSUB para comprobar si la barra espaciadora ha sido pulsada o no.

STRIG 1

```
10 SCREEN 0
20 ON STRIG GOSUB 70,90,110,130,150
30 FOR N=0 TO 4:STRIG(N) ON
40 STRIG(N) ON
50 NEXT N
60 GOTO 60
70 PRINT "Pulsado barra espaciadora"
80 RETURN
90 PRINT "Joystick 1, pulsado fuego 1"
100 RETURN
110 PRINT "Joystick 2, pulsado fuego 1"
120 RETURN
130 PRINT "Joystick 1, pulsado fuego 2"
140 RETURN
150 PRINT "Joystick 2, pulsado fuego 2"
160 RETURN
```

STRIG 2

```
10 SCREEN 0
20 PRINT "Pulse la barra espaciadora"
30 PRINT
40 A$=INKEY$
50 IF A$="" THEN GOTO 40
60 IF ASC(A$)=32 THEN PRINT "Pulsado bar
ra espaciadora"
70 GOTO 40
```

Comandos Gráficos PAD

Lee las coordenadas del pad conectado en el port de joystick

vea también los programas del apéndice: PAD 1
PAD 2
PAD 3

A diferencia de la mayor parte de ordenadores domésticos, los MSX no sólo disponen de dos conexiones para joystick, sino que además poseen una extensa variedad de comandos BASIC que permiten sacar un mayor provecho de las mismas.

PAD es uno de los mencionados comandos, y ofrece las entradas que se están efectuando en ese momento en el pad conectado. La lectura se realiza a través de una variable dimensionada del BASIC, designada igualmente con PAD.

A continuación, y antes de profundizar en el estudio de este comando BASIC, explicaremos brevemente el significado de un PAD y su uso.

En cierta medida, un PAD es una especie de bandeja cuya superficie se compone en gran parte de un material sensible a la presión. Con ayuda de un objeto puntiagudo podemos provocar señales en dicha superficie, que transmiten a través del port del joystick la información sobre el lugar donde se ha efectuado la presión. Imaginemos que la superficie del PAD sea una pantalla del ordenador. Si presionamos con un objeto puntiagudo en la esquina superior izquierda de la superficie del PAD, el ordenador registra esta señal en el lugar correspondiente y procesa los datos entrados en forma de sistema de coordenadas.

Es decir, por un lado el ordenador MSX verifica que queremos transmitir una acción presionando con un objeto; por otro lado puede determinar también las coordenadas exactas del lugar donde se efectuó la presión. El PAD dispone además de un botón (parecido al botón de fuego del joystick), que al pulsarlo provoca una determinada reacción en el ordenador. Ambos datos son registrados y podrán ser leídos con ayuda

del comando PAD del MSX.

El PAD puede utilizarse siempre y cuando se disponga de un programa confeccionado de tal forma (vea más adelante) que al presionar, por ejemplo, sobre un punto en la esquina superior izquierda del PAD, aparezca también un punto en la esquina superior izquierda de la pantalla, y que lo mismo ocurra para cualquier otro punto de la pantalla. Esto significa que podemos dibujar con ayuda de un objeto puntiagudo sobre el PAD, y que este dibujo aparecerá paralelamente en la pantalla.

Una buena y además útil aplicación del PAD es la siguiente: usted indica una determinada escala al ordenador, que relaciona una cierta longitud de línea con su correspondiente medida en kilómetros. Ahora coloque encima del PAD una porción de un mapa a la escala en cuestión, y verá cómo el ordenador calcula automáticamente las líneas dibujadas.

El botón adicional del PAD puede concebirse como un conmutador entre dos distintas imágenes/escalas o entre imagen y menú de ayuda. También puede utilizarse, por ejemplo, para cambiar el color del dibujo. Sin embargo, no olvidemos que el PAD sirve principalmente para entrar dibujos al ordenador y el botón para cambiar, por ejemplo, del menú a la pantalla (y viceversa).

Desgraciadamente, el ordenador MSX no dispone de un programa incorporado para controlar el PAD, de forma que debemos confeccionarlo nosotros mismos. Analicemos para ello la variable PAD: con ayuda de la instrucción PRINT PAD(N) podemos leer el movimiento de la variable del comando PAD, cuya asignación es la siguiente:

PAD(0) a PAD(3): lectura correspondiente al port 1
PAD(4) a PAD(7): lectura correspondiente al port 2

Cada variable PAD tiene un significado específico:

0 ó 4: lectura del pad conectado
1 ó 5: lectura de la coordenada X
2 ó 6: lectura de la coordenada Y
3 ó 7: lectura del botón adicional del PAD

En cualquier caso no basta con leer únicamente la variable PAD 0 ó 4, puesto que tan sólo indica si sucede alguna cosa en el PAD (resultado verdadero = -1), pero no especifica la forma y el lugar en que sucede.

Podemos concebir PAD(0) o PAD(4) como un interruptor que activa la lectura del PAD: si su resultado es -1, al mismo tiempo se modifica al menos una de las variables correspondientes a la entrada de coordenadas X-Y (PAD(1) a PAD(6)).

Al leer PAD(0) o PAD(4) mientras dibujamos en el PAD (presionando con un objeto puntiagudo sobre la bandeja), se emite el mensaje "verdadero = -1". En este caso (PRINT PAD(0): resultado = -1), PAD(1) y PAD(2) o bien PAD(5) y PAD(6) toman los actuales valores X,Y, como por ejemplo:

vea listado de PAD 1 del apéndice

Las últimas coordenadas indicadas se muestran en la pantalla hasta que se realice la próxima presión sobre el PAD (si es que se ha presionado = vea línea 40); por ejemplo, si presionamos con el objeto en el centro del PAD (128,96), en PAD(0) a PAD(2) ocurre lo siguiente:

1) PAD(0) tiene valor -1 (verdadero), mientras se mantenga presionado el lápiz.

2) PAD(1) toma el actual valor X (indicación de dicho valor mediante PRINT PAD(1))

3) PAD(2) toma el actual valor Y (indicación de dicho valor mediante PRINT PAD(2))

Por cierto, lo mismo también vale para cuando conectamos el

PAD en el port 2 del joystick. En este caso debemos efectuar la lectura a través de las variables PAD(4) (-1 = verdadero al tocar la superficie sensible del PAD con un objeto puntiagudo), PAD(5) y PAD(6) (coordenadas X-Y del punto de presión).

PAD(3) y PAD(7), correspondientes respectivamente al port 1 y port 2 del joystick, informan si ha sido pulsado o no el botón adicional del PAD (resultado afirmativo, o sea, verdadero = -1). Como ya mencionamos antes, tal indicación determina, por ejemplo, si deseamos cambiar a otra imagen.

A continuación analizaremos dichos resultados a través de un programa de ejemplo:

<u>vea listado de PAD 2 del apéndice</u>

Línea 30: la ejecución del programa se mantiene en esta línea hasta que el lápiz toque la superficie sensible del PAD (PAD(0) = 0; resultado falso), en caso contrario (PAD(0) = -1; resultado verdadero), el programa salta hacia la línea 40.

Línea 40: En la pantalla se emiten las coordenadas X e Y del punto donde el lápiz ha presionado sobre el PAD (por ejemplo, las coordenadas del centro: X(PAD(1)) = 128 e Y (PAD(2)) = 96)

Línea 50: Ahora el programa espera hasta que se pulse el botón del PAD (PAD(3) = verdadero = -1)

Línea 60: El programa salta hacia el principio.

El siguiente programa le mostrará una aplicación del PAD como instrumento de dibujo:

<u>vea listado de PAD 3 del apéndice</u>

Este programa no depende de comprobar en cada pasada si se ha tocado la superficie sensible del PAD, puesto que cada

vez se reactiva el actual cursor gráfico (línea 60). El peso del programa está más bien en leer el botón del PAD, que oscila entre el color del primer plano (negro) y el del fondo (blanco), según tecla pulsada. De esta forma, y con un poco de paciencia, se pueden dibujar cuadros directamente en el PAD y transmitirlos directamente a la pantalla. Incluso es posible volver a borrar superficies ya dibujadas de la pantalla, repintándolas con el color del fondo, tras cambiar a dicho color pulsando el botón del PAD.

Las coordenadas X-Y del PAD también pueden leerse directamente en las posiciones de memoria siguientes:

Coordenada X del PAD: &HFC9C
Coordenada Y del PAD: &HFC9D

PAD 1

```
10 COLOR 1,15
20 SCREEN 0
30 A=PAD(0)
40 IF A THEN GOTO 50 ELSE GOTO 30
50 PRINT PAD(1)
60 PRINT PAD(2)
70 GOTO 30
```

PAD 2

```
10 COLOR 1,15
20 SCREEN 0
30 IF PAD(0)=0 THEN GOTO 30
40 PRINT PAD(1),PAD)(2)
50 IF PAD(3)=0 THEN GOTO 50
60 GOTO 30
```

PAD 3

```
10 COLOR 1,15
20 SCREEN 0
30 IF PAD(3)=-1 THEN IF A=2 THEN A=1 ELS
E A=2
40 IF A=2 THEN COLOR 15
50 IF A=1 THEN COLOR 1
60 PSET(PAD(1),PAD(2))
70 GOTO 30
```

Comandos Gráficos PDL

Lee el paddle conectado en un port de joystick

vea también los programas del apéndice: PDL 1
PDL 2

A través del BASIC MSX disponemos de una gran variedad de comandos que nos permiten usar los ports de joystick para diversos periféricos. PAD nos ofrece las coordenadas X-Y del "touch pad" conectado al ordenador, STRIG ofrece las direcciones del joystick y STICK la pulsación/no pulsación de su botón de fuego. PDL sirve para leer otro periférico de control, el PaDdLe o "raquetas".

Mientras que la lectura del joystick ofrece una de las ocho posibles direcciones, la del paddle indica un punto del sistema de coordenadas de 256*192 puntos. Sin embargo, el control del paddle sólo se realiza en una dimensión (de derecha a izquierda o de arriba a abajo), en un ámbito de valores de 0 a 255.

El paddle es un instrumento de control que permite entrar valores entre 0 y 255 gracias a un botón giratorio. Recordemos, por ejemplo, un juego en el cual el paddle es ideal para dirigir la siguiente acción:

Nos encontramos en un circo y debemos mover un balancín apoyado en el suelo,de izquierda a derecha en la pantalla. Un muñequito salta de la cúpula del circo y usted debe colocarle el balancín de tal manera que su cuerpo adquiera la mayor altura de salto posible al caer encima de él (depende de si cae en el extremo de la parte levantada del balancín, con lo que el efecto de palanca es mayor, o cerca del centro giratorio).

Si quisiésemos escribir dicho programa para nuestro ordenador MSX, el siguiente subprograma bastaría para llevar el control a través de un paddle conectado:

vea listado PDL 1 del apéndice

Según la dirección de giro del paddle se generan valores entre 0 y 255 que permiten el control directo en horizontal de nuestro sprite en la pantalla.

Podríamos decir que PDL no es un comando sino una variable dimensionada. Sin embargo, esta variable se va actualizando continuamente según la posición del botón giratorio del paddle en cada momento.

No basta con leer simplemente la variable PDL; debemos asignarle un valor según las reglas válidas para los dos ports existentes de joystick o de paddle:

parámetro par (de 2 a 12) p.e. PDL(2) o PDL(4):
lectura del port 2
parámetro impar (de 1 a 11) p.e. PDL(1) o PDL(3):
lectura del port 1:

Aunque en este caso sólo utilizamos los valores 1 y 2, también es posible sustituirlos por ejemplo por 3 y 4, pero no deben exceder de los límites inferior y superior de 1 y 12, puesto que ello provocaría la interrupción del programa procesado en el ordenador, anunciado por el mensaje de error 'Illegal function call'.

Otras tres advertencias:

1) La lectura del port del joystick mediante PDL requiere unos 0.01 segundos. Es un tiempo bastante largo comparado con otros comandos del BASIC MSX. Por lo tanto, esta lectura de variable también puede servirle de bucle de espera incorporado en programas.

2) Si hay un paddle conectado a su ordenador MSX, el valor emitido en la lectura a través de la variable PDL es 255.

3) Puesto que un paddle siempre permanece en la última posición de la pantalla, que también podrá ser leída de nuevo directamente (mientras que el mando de un joystick

vuelve directamente a su original posición central en el momento de soltarlo, y el punto señalado en un "touch pad" sólo se memoriza mientras existe la presión sobre él), también podemos utilizar la lectura de 2 paddles conectados para un instrumento de dibujo:

vea listado de PDL 2 del apéndice

¡Pero cuidado! Aunque la extensión en dirección Y sobre la pantalla 'sólo' llega de 0 a 191, al leer el PDL(2) se emiten valores entre 0 y 255.

PDL 1

```
10 COLOR 1,15
20 SCREEN 2
30 SPRITE$(1)=STRING$(4,0)+CHR$(255)+CHR
$(255)+CHR$(24)+CHR$(24)
40 PUT SPRITE 1,(PDL(1),180)
50 GOTO 40
```

PDL 2

```
10 COLOR 1,15
20 SCREEN 2
30 PSET(PDL(1),PDL(2))
40 GOTO 30
```

Activa, desactiva e interrumpe el control de colisión entre sprites

Este comando del BASIC trabaja directamente junto con la instrucción ON SPRITE GOSUB. En este caso se trata de un control de interrupción, es decir, mientras se está ejecutando el programa en BASIC se va comprobando cada X unidades de tiempo (sin interrumpir la ejecución del programa) si se cumple cierta condición.

Aplicándolo al caso del comando ON SPRITE, esto significa comprobar si han colisionado 2 sprites en la pantalla. En caso afirmativo, el programa salta al subprograma indicado en la instrucción ON SPRITE GOSUB.

En el campo de los juegos encontramos ejemplos donde el 'control de colisión de sprites' puede aplicarse de forma útil:

Por ejemplo, dos naves colisionan entre sí. El subprograma visualiza una explosión en la correspondiente posición de la pantalla.

O un jugador dispara sobre objetos voladores. En el momento en que su munición (un sprite) choca contra un objeto (otro sprite), el ordenador va hacia un subprograma, incrementa la puntuación del jugador en 1 y borra el objeto tocado de la pantalla.

¿Para qué sirve entonces el comando SPRITE ON/OFF/STOP? Sirve para interrumpir (mediante SPRITE OFF) el salto previamente definido hacia el subprograma -el número de línea de este subprograma ha sido indicado en la instrucción ON SPRITE GOSUB- según la situación, volver a ejecutarlo de nuevo (mediante SPRITE ON) o provocar su ejecución en un momento posterior tras un nuevo comando SPRITE ON (mediante SPRITE STOP).

Nuestro juego podría estar pensado de forma que al principio cada colisión con algún ser provocase la destrucción del jugador. Superada la primera vuelta, la situación cambia: ahora se trata de colisionar con el mayor número posible de seres, obteniendo un bono por cada colisión. En esta situación es absurdo que continúen produciéndose explosiones como en la primera fase del juego. Por ello nos servimos del comando SPRITE OFF para interrumpir el salto hacia el correspondiente subprograma. Si a continuación el juego vuelve a su estado inicial, podemos comunicárselo al ordenador mediante el comando SPRITE ON.

Para comprobar que dos sprites han colisionado, no hace falta que se superpongan los puntos visualizados de los sprites. El ordenador ya detecta tal colisión cuando las matrices de dos sprites se solapen en un único punto. Por lo tanto, puede ocurrir que el ordenador comunique la superposición de dos sprites cuando ésta no se perciba visiblemente en la pantalla (en este caso, los elementos de sprites no ocupados con puntos no se han solapado de forma visible para nosotros).

En otro caso es imposible controlar la colisión entre sprites a través del comando SPRITE ON: Si trabaja con sprites de diversos colores, el ordenador MSX emite continuamente el mensaje de colisión. Por lo tanto, deberá decidir entre la comprobación de colisión de sprites o los sprites de diversos colores, puesto que ambas condiciones juntas son incompatibles.

Un último punto que exige especial precaución: cuando el ordenador recibe el mensaje de la colisión, debemos finalizar de momento la comprobación de tal suceso mediante el comando SPRITE OFF, para evitar que los dos sprites, cuyo choque ha sido registrado como tal, vayan aproximándose hasta solaparse completamente. En este caso, el ordenador recibiría continuamente la información respecto al choque producido, cosa que limitaría considerablemente la ejecución del programa.

El ordenador MSX sólo reconoce que dos sprites chocan entre sí, pero no sabe de qué sprites se trata ni en qué lugar se encuentran. No es fácil obtener esta información, aunque en juegos sea importante conocerla (no importa si chocan dos naves del mismo jugador, pero la cosa cambia si se trata de dos objetos de jugadores diferentes). ¿Cómo podemos conocer los sprites que han colisionado?

Por un lado, en la tabla de atributos de sprites pueden comprobarse las coordenadas del punto donde se ha producido la colisión. Sin embargo, este procedimiento es muy largo y engorroso, por lo que no interesa en juegos rápidos. Pero existe otro procedimiento más rápido y sencillo para conocer los sprites que han colisionado en la pantalla:

Durante la ejecución del juego, y con cada movimiento de cualquier sprite, los datos sobre el sprite que acaba de ser desplazado van asignándose a una variable. Gracias a esta información podremos conocer inmediatamente el causante de la colisión. El segundo sprite (objeto pasivo del choque) debe encontrarse dentro de la matriz de puntos del sprite activo en cuestión. A tal fin consulte la VRAM con ayuda de VPEEK y sabrá rápidamente cuál es el sprite que ha recibido el choque, al que se restarán puntos (porque ha sido tocado por el contrario) o bien se le sumarán puntos (porque no es el causante de la colisión) según el objetivo del juego.

Finalmente, ¿cómo podemos averiguar si se trata del sprite de uno u otro jugador? Existen dos posibilidades de diferenciarlos: definir ambos bandos de forma que el jugador 1 posea los sprites de números impar (1,3,5, ...) y el jugador 2 los de número par (2,4,6, ...), o bien el jugador 1 dispone de los sprites 1 a 4, mientras que el jugador 2 obtiene los restantes de 5 a 8. También es posible asignar los sprites de forma desordenada a los jugadores (sin tener en cuenta la mayor complicación para programar el control de los sprites), y comprobar en la VRAM (tabla de atributos de sprites) el color del jugador cuando se produzca una superposición de figuras móviles.

Finalizaremos con una última advertencia: cada comando BASIC obtiene sus datos de otros datos almacenados en alguna parte del ordenador. El mensaje de colisión de dos sprites también se efectúa de igual forma: si leemos los ocho registros del procesador de vídeo mediante .

PRINT VDP(8)

obtendremos los datos sobre la superposición de dos sprites, aunque previamente hayamos entrado el comando SPRITE OFF. El resultado emitido sobre el bit 5 del registro 8 del VDP será 1 (verdadero) en caso de accidente y 0 (falso) en el caso contrario. Sin embargo, es aconsejable utilizar esta lectura únicamente si con ella podemos provocar/interrumpir una reacción a la interrupción razonable/irrazonable del programa. De lo contrario, su programa se volvería demasiado lento comparado con la cómoda lectura de interrupción efectuada mediante el comando ON SPRITE o SPRITE ON/OFF.

Debe ser consciente de que al leer el registro 8 se desactiva automáticamente el control de colisión de sprites (correspondiente al comando SPRITE OFF). Por lo tanto, debe entrar de nuevo el comando SPRITE ON -si lo desea- después de cada lectura del registro 8 del VDP.

Comandos Gráficos STRIG(N) ON/OFF/STOP

Activa y desactiva el control de interrupciones ON STRIG GOSUB

vea también los programas del apéndice: STRIG(N) ON 1
STRIG(N) ON 2
STRIG(N) ON 3
STRIG(N) ON 4
STRIG 1
STRIG 2
MISTER MINER
FRETY

Un especialidad del BASIC MSX consiste en permitir que, aun sin tener conocimientos del lenguaje máquina, se introduzca dicho lenguaje durante la ejecución de programas BASIC indirectamente a través de rutinas de interrupciones. Los comandos encargados de ello comienzan con el término ON e indican que, tras definirlo una sola vez, este control será ejecutado automáticamente por el ordenador MSX mientras nosotros se lo permitamos (ON ERROR = si aparece un error; ON INTERVAL = en determinados intervalos de tiempo; ON KEY = si se ha pulsado determinada tecla de función; etc.)

El interruptor de estas rutinas indica que tras un ON se inicie el control independiente del ordenador, y que un OFF provoque la interrupción de la lectura y su consiguiente reacción, mientras que un STOP no interrumpe el control pero sí cualquier reacción posterior (bifurcación hacia un subprograma). Al encontrarse otro STRIG(N) ON, se percibe el último joystick pulsado durante STRIG(N) STOP (si realmente ha sucedido así) y, a continuación, se salta hacia el correspondiente subprograma. Esto tiene el sentido de evitar una nueva bifurcación a partir de una subrutina de joystick que se está ejecutando (con STRIG(N) ON) hacia otra subrutina de joystick sin haber procesado las líneas. Es conveniente efectuar la segunda entrada STRIG(N) ON después de finalizar el proceso del primer subprograma, recuperando de esta forma la información almacenada durante la fase de STRIG(N) STOP.

vea listado de STRIG(N) ON 1 del apéndice

El comando STRIG(N) ON/OFF/STOP hace referencia a la rutina de interrupción que controla la pulsación de los botones de joysticks y de la barra espaciadora. Sin embargo, las interrupciones únicamente pueden provocar reacciones si se cumple lo siguiente:

barra espaciadora pulsada: STRIG(0) ON

joystick 1, botón de fuego 1 pulsado: STRIG(1) ON

joystick 2, botón de fuego 1 pulsado: STRIG(2) ON

joystick 1, botón de fuego 2 pulsado: STRIG(3) ON

joystick 2, botón de fuego 2 pulsado: STRIG(4) ON

La lectura de los botones especiales de joystick para MSX (2 botones de fuego de diferentes funciones por cada joystick) sólo debería activarse mediante STRIG(3) ON o STRIG(4) ON en el caso de poseer realmente un joystick MSX.

¿Para qué sirve activar y desactivar la lectura de los botones disparadores de los joysticks?

Si trabajamos con joysticks (y no con otros instrumentos de control que pueden conectarse igualmente en los ports de joysticks del ordenador MSX), el comando STRIG(N) ON/OFF/STOP suele aplicarse en los juegos. Vea los tres ejemplos siguientes:

1) En una primera fase de juego desea evitar que el jugador 1 pueda disparar varios tiros uno tras otro en intervalos cortos de tiempo (quizás porque ello provacaría la ejecución demasiado lenta de su programa). Por lo tanto, en posición de STRIG(N) ON espera a que se produzca un disparo, después entra el comando STRIG(N) OFF y a continuación sigue el curso de la flecha. Cuando la flecha desaparezca de la

ventana de la pantalla podrá efectuarse otro disparo (entrar de nuevo STRIG(N) ON).

vea listado de STRIG(N) ON 2 del apéndice

Para conocer la diferencia entre STRIG(N) ON, STRIG(N) OFF y STRIG(N) STOP, modifique cada vez la línea 110 de este programa y a continuación, mientras aparezca la flecha en la pantalla, vaya pulsando continuamente la barra espaciadora:

Con STRIG(N) ON se encuentran finalmente múltiples flechas o partes de flechas en la pantalla, puesto que se va bifurcando siempre de nuevo a partir de la subrutina para visualizar otra flecha;

con STRIG(N) OFF no ocurre nada especial y

con STRIG(N) STOP, el ordenador memoriza la pulsación de la tecla o del botón del joystick y ejecuta la función cuando la imagen visualizada de la flecha haya desaparecido por la parte superior de la pantalla.

2) Otro juego exige que la pulsación del botón de fuego del joystick se efectúe en el momento en que se emita un sonido determinado. Para este cometido nos servimos de nuevo del comando STRIG(N) ON/OFF/STOP:

vea listado de STRIG(N) ON 3 del apéndice

3) Dos jugadores pueden pulsar alternativamente el botón de fuego de su joystick. Con ayuda de los comandos STRIG(1) ON/OFF y STRIG(2) ON/OFF permitimos dicha pulsación alternativamente al jugador 1 y al jugador 2, evitando al mismo tiempo que el jugador contrario pueda pulsar el botón.

vea listado de STRIG(N) ON 4 del apéndice

STRIG(N) ON 1

```
10 SCREEN 0
20 PRINT "Pulse la barra espaciadora"
30 PRINT "durante la emision de numeros"
40 FOR N=1 TO 1000
50 NEXT N
60 ON STRIG GOSUB 140
70 STRIG(O) ON
80 STRIG(O) STOP
90 FOR N=1 TO 100
100 PRINT N
110 NEXT N
120 STRIG(O) ON
130 GOTO 130
140 PRINT "Barra espaciadora pulsada"
```

STRIG(N) ON 2

```
10 SCREEN 0
20 COLOR 1,15
30 PRINT "Pulse la barra espaciadora"
40 PRINT "para disparar"
50 FOR N=1 TO 1000
60 NEXT N
70 SCREEN 2
80 ON STRIG GOSUB 110
90 STRIG(O) ON
100 GOTO 100
110 STRIG(O) OFF
120 FOR N=191 TO 1 STEP -2
130 DRAW "c1 bm 128,=n; U8 N F4 G4"
140 DRAW "c15 bm 128,=n; U8 N F4 G4"
150 NEXT N
160 STRIG(O) ON
170 RETURN
```

STRIG(N) ON 3

```
10 SCREEN 0
20 PRINT "Pulse la barra espaciadora"
30 PRINT "cuando suene la nota DO"
40 PRINT
50 ON STRIG GOSUB 160
60 STRIG(O) ON
70 FOR N=1 TO 10
80 READ A$
90 IF A$="C" THEN STRIG(O) ON ELSE STRIG
(O) OFF
```

```
100 PLAY "11"+A$
110 IF PLAY(1) THEN GOTO 110
120 NEXT N
130 PRINT "de 3 posibilidades:":Z
140 END
150 DATA C,E,F,C,D,F,A,C,D,E
160 Z=Z+1
170 STRIG(0) OFF
180 RETURN
```

STRIG(N) ON 4

```
10 SCREEN 0
20 COLOR 1,15
30 ON STRIG GOSUB ,240,240
40 PRINT "Jugador 1 puede disparar"
50 PRINT "ahora con joystick 1"
60 FOR N=1 TO 1000
70 NEXT N
80 STRIG(2) OFF
90 STRIG(1) ON
100 SCREEN 2
110 FOR N=1 TO 1000
120 NEXT N
130 SCREEN 0
140 PRINT "Jugador 2 puede disparar"
150 PRINT "ahora con joystick 2"
160 FOR N=1 TO 1000
170 NEXT N
180 STRIG(2) ON
190 STRIG(1) OFF
200 SCREEN 2
210 FOR N=1 TO 1000
220 NEXT N
230 RUN
240 STRIG(1) OFF
250 STRIG(2) OFF
260 FOR M=191 TO 0 STEP-2
270 DRAW "c1 bm 128,=m; U16 N F4 G4"
280 DRAW "c15 bm 128,=m; U16 N F4 G4"
290 NEXT M
300 RETURN
```

Comandos   Gráficos   ON SPRITE GOSUB

Controla la colisión de sprites a través de interrupciones

vea también los programas del apéndice: ON SPRITE 1
ON SPRITE 2

El ordenador MSX permite, durante la ejecución de un programa BASIC, ir comprobando continuamente si ha ocurrido algo a lo que debe reaccionar de alguna manera. Los comandos que posibilitan dicha acción son ON INTERVAL y ON KEY GOSUB.

Utilizando el comando ON SPRITE ordenamos que el ordenador vaya averiguando si en algún momento durante la ejecución del programa se produce una colisión entre dos sprites. Cuando ello ocurra, deberá saltar en un subprograma hacia una línea indicada para este caso. Tal como se señala en el siguiente programa, al igual que los dos programas de juego incorporados en este libro, el comando ON SPRITE GOSUB tiene su principal aplicación en los juegos: permite averiguar, por ejemplo, si colisionan dos cohetes, como verá en el siguiente programa:

vea listado de ON SPRITE 1 del apéndice

Cada vez que se superponen dos sprites en el centro de la pantalla suenan 4 pitidos consecutivos. La función ON SPRITE y el siguiente 'comando de activación' (SPRITE ON) de las líneas 30 y 40 provocan que el ordenador controle a partir de ese momento si en algún lugar se superponen 2 sprites y que, en caso necesario, salte hacia el subprograma de la línea 120.

Podemos desactivar el comando ON SPRITE en cualquier momento mediante SPRITE OFF; también podemos continuar controlando la colisión de sprites, pero sin efectuar ninguna reacción (salto a subprograma) en consecuencia (en primer lugar SPRITE STOP, más tarde SPRITE ON; con ello conseguimos que se dirija ulteriormente al correspondiente subprograma. Consulte también los programas de ejemplo del capítulo SPRITE ON).

Es posible leer qué sprite ha chocado contra qué sprite. Para ello debemos almacenar cada movimiento de sprites en una memoria intermedia, tal como ha ocurrido en el siguiente programa y como se muestra en la pantalla:

vea listado de ON SPRITE 2 del apéndice

Es decisivo que los valores 1 (para sprite 1) ó 2 (para sprite 2) sean asignados a la variable Z antes de ejecutar el propio comando del movimiento de sprites, para evitar que el ordenador se bifurque hacia el subprograma, antes de haber obtenido la información para saber cuál de los prites ha sido el causante de la colisión.

En este programa además es aconsejable (y por ello se procede de esta forma) desconectar la rutina ON SPRITE mediante SPRITE OFF tras ejecutar una vez el bucle del subprograma. ¡Pruebe lo que ocurre en el caso contrario!

Los sprites vuelven a superponerse otra vez al efectuar el siguiente pequeño movimiento, y después otras dos veces. Si mientras tanto no se efectúa ningún comando SPRITE OFF, el ordenador volverá a bifurcarse continuamente hacia el subprograma de la línea 140, sin finalizar cada vez el proceso del mismo.

De la indicación visualizada en la pantalla puede deducir, que el choque (línea 110) es provocado continuamente de nuevo por el sprite 2. Respecto a ello le damos la siguiente información importante:

El comando ON SPRITE GOSUB siempre tiene efecto cuando los sprites se superponen de alguna forma en algún lugar del gráfico. Si tenemos dos muñequitos dibujados en forma de 2 sprites puede ocurrir fácilmente, que la ejecución de la rutina ON SPRITE sea provocada no por la superposición sino por el simple contacto cercano. Para evitar tal circunstancia se precisa de un enorme subprograma. De momento puede recurrir a la solución de hacer que los

muñequitos no se muevan punto por punto, sino paso por paso (varios puntos seguidos); ello provoca que los movimientos se realicen a golpes, pero en contrapartida se consigue una ejecución precisa del comando ON SPRITE.

Por cierto:

1) Queremos insistir especialmente en lo siguiente: no olvide nunca desactivar el comando de colisión SPRITE mediante SPRITE OFF, si quiere evitar los continuos saltos hacia el subprograma.

2) Puede leer el registro de colisiones de sprites utilizando el comando VDP (bit 5 del registro 8) o comprobando la variable del sistema a partir de &HF3DF. Para mayor información al respecto consulte la descripción del comando VDP.

ON SPRITE 1

```
10 COLOR 1,15
20 SCREEN 2
30 ON SPRITE GOSUB 130
40 SPRITE ON
50 SPRITE$(1)="UUUUUUUU"
60 SPRITE$(2)="UUUUUUUU"
70 FOR N=1 TO 255
80 PUT SPRITE 1,(N,96)
90 PUT SPRITE 2,(256-N,96)
100 NEXT N
110 SPRITE ON
120 GOTO 60
130 SPRITE OFF
140 BEEP
150 RETURN
```

ON SPRITE 2

```
10 COLOR 1,15
20 SCREEN 1
30 ON SPRITE GOSUB 150
40 SPRITE ON
50 SPRITE$(1)="UUUUUUUU"
60 SPRITE$(2)="UUUUUUUU"
70 FOR N=1 TO 255
80 Z=1
90 PUT SPRITE 1,(N,96)
100 Z=2
110 PUT SPRITE 2,(256-N,96)
120 NEXT N
130 SPRITE ON
140 GOTO 60
150 SPRITE OFF
160 PRINT Z;
170 CLOSE
180 SPRITE OFF
190 RETURN
```

Comandos Gráficos ON STRIG GOSUB
Controla el salto hacia el subprograma al pulsar el botón del joystick

vea también los programas del apéndice: ON STRIG 1
ON STRIG 2
LOCATE 9

Al igual que los comandos ON SPRITE GOSUB

vea listados de ON SPRITE 1 y 2 del apéndice

y ON INTERVAL GOSUB

vea listado de LOCATE 9 del apéndice

explicados anteriormente, ON STRIG GOSUB también es un comando del MSX que permite que el propio lenguaje de programación BASIC trabaje directamente junto con el lenguaje máquina.

En este caso, el programa BASIC normal va ejecutándose hasta que se cumpla la condición formulada en la instrucción ON. Al controlar el programa máquina mediante ON ... GOSUB podemos evitarnos tener que comprobar sin interrupciones, si un suceso ha acaecido o no (por ejemplo, si el joystick ha sido pulsado o si dos sprites han colisionado), siendo el mismo ordenador MSX el que lo efectúe automáticamente.

¿Qué significa el comando ON STRIG GOSUB? Sirve para consultar al ordenador si se ha pulsado un botón de fuego del joystick o la barra espaciadora:

vea listado de ON STRIG 1 del apéndice

En cualquier caso, para conseguir alguna reacción es necesario poner en marcha el control en lenguaje máquina efectuado mediante ON STRIG, utilizando el comando STRIG(N) ON. Sin embargo, si lo desea (si se desea evitar la pulsación del botón de fuego en un juego quizás porque uno

mismo ya haya sido derribado) también puede interrumpir la lectura automática del botón del fuego o de la barra espaciadora, dando la instrucción STRIG(N) OFF. Finalmente, el comando STRIG(N) STOP provoca que el ordenador memorice la pulsación de los botones de fuego, pero su reacción se produzca después de ordenar nuevamente STRIG(N) ON.

Observemos los diversos estados que pueden indicarse a través de la lectura:

STRIG(0): barra espaciadora pulsada si/no
STRIG(1): botón de fuego 1 del joystick 1 pulsado si/no
STRIG(2): botón de fuego 1 del joystick 2 pulsado si/no
STRIG(3): botón de fuego 2 del joystick 1 pulsado si/no
STRIG(4): botón de fuego 2 del joystick 2 pulsado si/no

De esta forma, en la línea 40 podríamos bifurcar el programa hacia otro subprograma según la tecla pulsada, indicando los diversos números de línea correspondientes separados por comas:

vea listado de ON STRIG 2a del apéndice

Al pulsar, por ejemplo, la barra espaciadora, el programa se bifurca hacia la línea 1000; al pulsar el botón de fuego del joystick 2, la ejecución del programa continúa en la línea 5000; etc.

Para controlar realmente todos los botones de fuego de los joysticks, deben añadirse las líneas 40 a 70:

vea listado de ON STRIG 2b del apéndice

Si además deseamos ampliar un poco nuestro programa, la lectura de los botones de fuego es ideal para cumplir nuestro objetivo y ofrece una respuesta a todas las posibles preguntas:

vea listado de ON STRIG 2c del apéndice

Ahora puede comprobar si todos sus joysticks son compatibles con el MSX. En caso afirmativo, ambos botones de fuego (si existen) deberían generar respuestas diferentes en la pantalla:

1- joystick 1 botón de fuego 1 pulsado
2- joystick 2 botón de fuego 1 pulsado
3- joystick 1 botón de fuego 2 pulsado
4- joystick 2 botón de fuego 2 pulsado

Si conviene que cada jugador tenga las mismas posibilidades de pulsar botones, puede evitarse con STRIG(N) OFF que continúe participando en el juego.

Ahora compruebe en sus propios programas confeccionados hasta qué límites pueden utilizarse estos comandos (¿quién habló de niñerías?), o consulte los listados de los juegos incluídos en este libro.

ON STRIG 1

```
10 COLOR 1,15
20 SCREEN 0
30 ON STRIG GOSUB 60
40 STRIG(0) ON
50 GOTO 50
60 PRINT "Pulsado barra espaciadora"
70 RETURN
```

ON STRIG 2a

```
10 COLOR 1,15
20 SCREEN 0
30 ON STRIG GOSUB 1000,2000,3000,4000,50
00
```

ON STRIG 2b

```
40 FOR N=0 TO 4
50 STRIG(N) ON
60 NEXT N
```

ON STRIG 2c

```
999 GOTO 999
1000 PRINT "Barra espaciadora pulsada"
1010 RETURN
2000 PRINT "Joystick 1, boton fuego 1 pu
lsado"
2010 RETURN
3000 PRINT "Joystick 2, boton fuego 1 pu
lsado"
3010 RETURN
4000 PRINT "Joystick 1, boton fuego 2 pu
lsado"
4010 RETURN
5000 PRINT "Joystick 2, boton fuego 2 pu
lsado"
5010 RETURN
```

Descripción de programa 1

La RAM de vídeo queda borrada al entrar un comando SCREEN con el fin de cambiar de pantalla, por ejemplo, para saltar del modo normal de texto (SCREEN 0 ó SCREEN 1) a un modo gráfico (SCREEN 2 ó SCREEN 3).

La tabla de generadores de sprites queda borrada de la RAM de vídeo al modificar el segundo parámetro del comando SCREEN (por ejemplo, SCREEN ,1).

Como consecuencia de ello, cada vez que modifique el tamaño de los sprites deberá definir de nuevo todos los patrones utilizados anteriormente, o bien volver a leerlos de nuevo en líneas DATA.

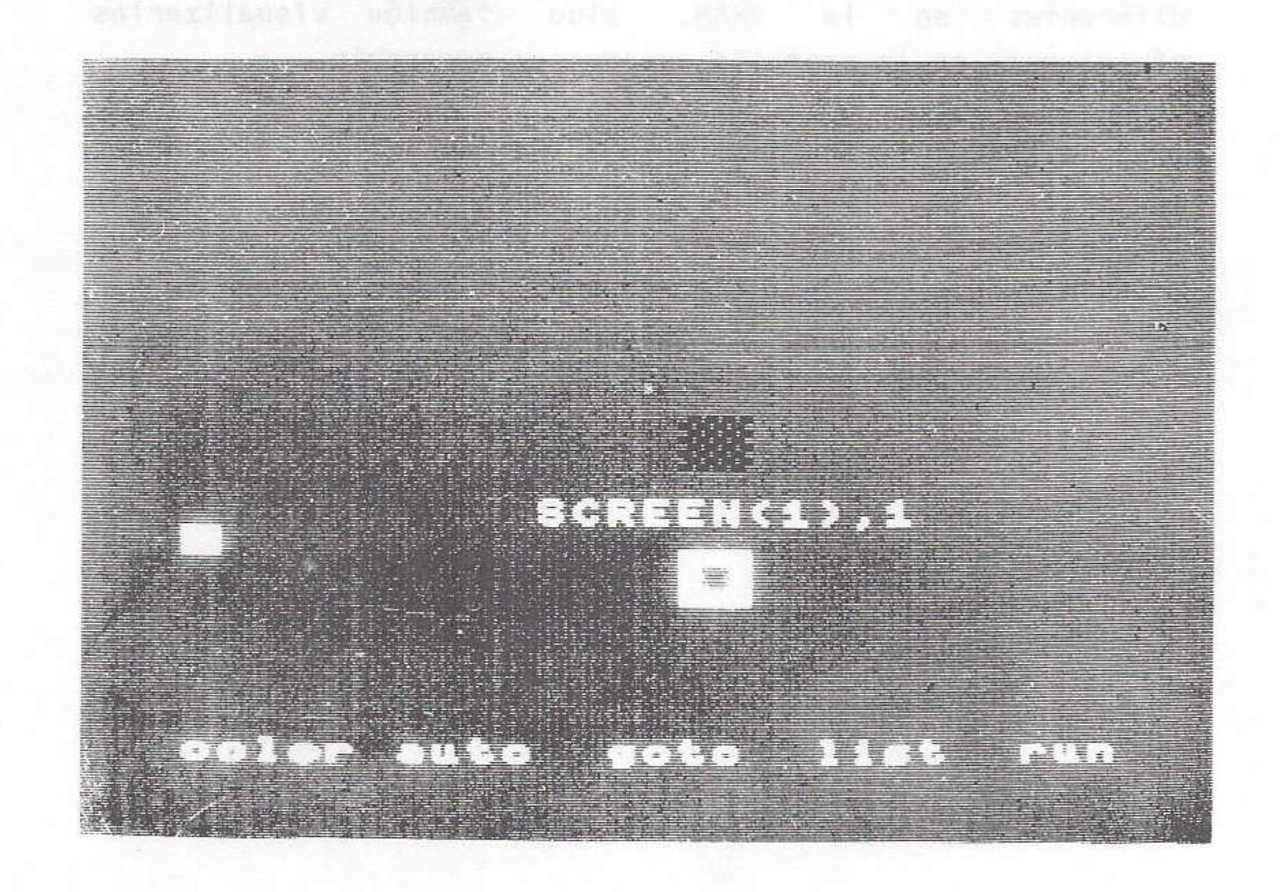
El comando VDP le ofrece la posibilidad de modificar el tamaño de los sprites durante la ejecución de un programa. De esta forma se evita el borrado de los patrones de sprites definidos anteriormente en la VRAM.

El programa 1 muestra de forma efectiva la aplicación del comando VDP en este sentido: en primer lugar se representan sprites de matriz 8*8 en su tamaño normal (un punto del sprite corresponde a un punto de la pantalla = SCREEN ,0) y después a escala ampliada (un punto del sprite corresponde a 4 puntos de la pantalla = SCREEN ,1). El patrón del sprite mantiene su aspecto inalterable al aumentar su tamaño, es decir, la carga del patrón sólo se efectúa una vez al principio del programa.

Si tiene dificultades en representar sprites, este programa le ayudará seguramente a solventarlos. Sin embargo, sería ideal poder representar en la pantalla todos los tamaños posibles.

```
10 REM Programa 1
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                    Rainer Luers
40 REM Sprites 8*8 normal y
       ampliado, alternando su
       representacion sin borrar VRAM
50 SCREEN 1
60 COLOR 15,4
70 REM Sprite 1 = tablero de ajedrez
80 DATA 01010101
90 DATA 10101010
100 DATA 01010101
110 DATA 10101010
120 DATA 01010101
130 DATA 10101010
140 DATA 01010101
150 DATA 10101010
160 REM Sprite 2 = borde
170 DATA 11111111
180 DATA 11111111
190 DATA 11000011
200 DATA 11000011
210 DATA 11000011
220 DATA 11000011
230 DATA 11111111
240 DATA 11111111
250 REM Lectura de los sprites 0 y 1
260 RESTORE 80
270 FOR M=0 TO 1
280 B$=""
290 FOR N=0 TO 7
300 READ A$
310 REM Cambio representacion binaria
                caracteres
320 B$=B$+CHR$(VAL("&b"+A$))
330 NEXT N
```

```
340 SPRITE$(M)=B$
350 NEXT M
360 FOR A=1 TO 100
370 NEXT A
380 REM Representacion sprites 0 y 1
390 PUT SPRITE 0,(128,86),1,0
400 PUT SPRITE 1,(128,126),15,1
410 REM Lectura del bit 0
        en registro 1 de VDP
420 B=VDP(1) AND 1
430 REM Cuando bit 0 = 1,
        entonces bit 0 = 0
440 IF B=1 THEN VDP(1)=VDP(1) AND &B1111
1110:LOCATE 10,14:PRINT "SCREEN(1),0":GO
TO 360
450 REM Cuando bit 0 = 0,
        entonces bit 0 = 1
460 IF B=0 THEN VDP(1)=VDP(1) OR &B00000
001:LOCATE 10,14:PRINT "SCREEN(1),1":GOT
O 360
```



Descripción de programa 2

En SCREEN 0 tiene la posibilidad de visualizar o almacenar hasta 960 caracteres (24 líneas por 40 caracteres) al mismo tiempo en la pantalla (y con ello también en la VRAM).

En SCREEN 0, la VRAM (más de 16000 posiciones de memoria) sólo es aprovechada parcialmente con estos 960 caracteres y sus aspectos (2048 posiciones de memoria). Las posiciones libres podrían utilizarse, por ejemplo, para almacenar variables a través de VPEEK y VPOKE o el comando SPRITE$; un objetivo poco deseable.

El programa 2 aprovecha mejor la VRAM al trabajar en SCREEN 0, almacenando en la memoria varias pantallas de texto al mismo tiempo. Este programa muestra que el comando VDP no sólo permite almacenar hasta 13 pantallas de texto diferentes en la VRAM, sino también visualizarlas rápidamente en la pantalla cuando sea necesario.

¿Qué sentido tiene todo esto? En estas 13 diferentes pantallas de texto podría almacenar, por ejemplo, informaciones auxiliares o fórmulas utilizadas frecuentemente. Su fantasía no está limitada para aprovechar estos cambios ultrarrápidos entre pantallas (por ejemplo, podría crear unos dibujos animados en las 13 pantallas).

```
10 REM Programa 2
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                        Rainer Luers
40 REM En SCREEN 0 trabajamos
       simultaneamente con hasta
       13 pantallas independientes
50 SCREEN 0
60 WIDTH 39
70 KEY OFF
80 REM Volver siempre a la pantalla de
       entrada tras interrumpir el
       programa
90 ON STOP GOSUB 430
100 STOP ON
110 M1=200
120 REM Las pantallas 1 a 12 obtienen
        una indicacion
130 FOR N=1 TO 12
140 PRINT "Inicializacion de la pantalla
";N
150 A$="Esta es la pantalla"+STR$(N)+STR
ING$(10,32)
160 REM POKEar la indicacion (A$) a
        la correspondiente posicion
        inicial de las 12 pantallas
170 FOR M=1 TO LEN(A$)
180 VPOKE (3+N)*1024+M,ASC(MID$(A$,M,1))
190 NEXT M
200 NEXT N
210 CLS
220 PRINT "Esta es la pantalla 0"
230 VDP(2)=0
240 REM Saltar al bucle de tiempo si se
        desea
250 GOSUB 390
260 REM Ojear las pantallas 1 a 12
```

```
270 FOR N=4 TO 15
280 VDP(2)=N
290 REM Saltar al bucle de tiempo si se
        desea
300 GOSUB 390
310 NEXT N
320 VDP(2)=0
330 PRINT
340 B$=""
350 INPUT "Indique la velocidad, por vav
or (1000 bis 1) ";B$
360 M1=VAL(B$)
370 GOTO 230
380 REM Bucle de tiempo
390 FOR M=1 TO M1
400 NEXT M
410 RETURN
420 REM Vuelta a la pantalla de entrada
        y fin del programa al pulsar
        las teclas CTRL STOP
430 VDP(2)=0
440 END
```

```
Inicializacion de la pantalla 1
Inicializacion de la pantalla 2
Inicializacion de la pantalla 3
Inicializacion de la pantalla 4
Inicializacion de la pantalla 5
Inicializacion de la pantalla 6
Inicializacion de la pantalla 7
Inicializacion de la pantalla 8
Inicializacion de la pantalla 9
Inicializacion de la pantalla 10
Inicializacion de la pantalla 11
Inicializacion de la pantalla 12
```

Descripción de programa 3

Trabajando en los modos de texto SCREEN 0 ó SCREEN 1 del ordenador MSX, tiene la posibilidad de representar simultáneamente hasta 256 caracteres diferentes.

El siguiente programa nos muestra parte de estos caracteres (aquellos que pueden ser visualizados en la pantalla):

```
10 FOR N=32 TO 255
20 PRINT CHR$(N);
30 NEXT N
```

Si modifica el aspecto de un carácter, al mismo tiempo se modificará también en la pantalla el aspecto de todos los caracteres que tengan el mismo código (consulte los capítulos CHR$ y ASC).

El programa 3 tampoco es capaz de representar más de 256 caracteres diferentes. Sin embargo ofrece la posibilidad, además de almacenar dos juegos de caracteres diferentes en la VRAM, de poder intercambiarlos también en fracciones de segundo. Para ello utilizamos nuevamente el comando VDP.

Para abreviar el programa he escogido el estándar en lugar del segundo juego de caracteres, convirtiendo sin embargo las letras mayúsculas en caracteres inversos.

```
10 REM Programa 3
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                         Rainer Luers
40 REM Programa de aplicacion de 2
       juegos de caracteres diferentes,
       que se pueden cambiar facil y
       rapidamente mediante VDP(4)
50 SCREEN 0
60 WIDTH 40
70 PRINT "Se almacena un segundo juego d
e         caracteres de mayusculas inver
tidas"
80 PRINT
90 REM Los caracteres 0 a 64 se
       transmiten con igual aspecto al
       segundo juego de caracteres
100 FOR N=0 TO 64
110 PRINT N;
120 FOR M=1 TO 8
130 GOSUB 660
140 NEXT M
150 NEXT N
160 REM Los caracteres 65 a 90
        (mayusculas) son calculados
        para el segundo juego de
        caracteres, adquiriendo asi
        un aspecto invertido
170 FOR N=65 TO 90
180 PRINT N;
190 FOR M=1 TO 8
200 A$=BIN$(VPEEK(2047+(N*8)+M))
210 A$=STRING$(8-LEN(A$),"0")+A$
220 REM Invertir configuracion binaria
        (es decir, donde hay un 0 habra
         un 1 y viceversa)
```

```
230 FOR L=1 TO 8
240 IF MID$(A$,L,1)="0" THEN MID$(A$,L,1
)="1" ELSE MID$(A$,L,1)="0"
250 NEXT L
260 VPOKE 4095+(N*8)+M,VAL("&B"+A$)
270 NEXT M
280 NEXT N
290 REM Los caracteres 91 a 254 se
        transmiten con igual aspecto al
        segundo juego de caracteres
300 FOR N=91 TO 254
310 PRINT N;
320 FOR M=1 TO 8
330 GOSUB 660
340 NEXT M
350 NEXT N
360 REM El caracter del cursor se
        convierte en tablero de ajedrez
370 FOR M=1 TO 8 STEP 2
380 VPOKE 4095+(255*8)+M,85
390 VPOKE 4095+(255*8)+M+1,170
400 NEXT M
410 CLS
420 PRINT "Actualmente su juego de carac
teres es:"
430 PRINT
440 REM Impresion del juego de carac-
        teres visible en la pantalla en
        este momento
450 FOR N=32 TO 255
460 PRINT CHR$(N);
470 NEXT N
480 PRINT
490 PRINT
500 GOSUB 690
510 PRINT "Ahora cambiamos al otro juego
 de          caracteres (MAYUSCULAS INVERT
IDAS)        con VDP(4)=2"
```

```
520 PRINT
530 GOSUB 690
540 REM Cambio a juego de caracteres 2
550 VDP(4)=2
560 PRINT "Ahora efectuamos 1000 cambios
 rapidos   entre los 2 juegos de caracte
res"
570 GOSUB 690
580 FOR N=1 TO 1000
590 REM Llamada juego de caracteres 1
600 VDP(4)=1
610 REM Llamada juego de caracteres 2
620 VDP(4)=2
630 NEXT N
640 END
650 REM Copiar juego de caracteres
660 VPOKE 4095+(N*8)+M,VPEEK(2047+(N*8)+
M)
670 RETURN
680 REM Bucle de espera
690 FOR N=1 TO 1000
700 NEXT N
710 RETURN
```

```
Se almacena un segundo juego de
caracteres de mayusculas invertidas

0   1   2   3   4   5   6   7   8   9   10   11
12   13   14   15   16   17   18   19   20   21
22   23   24   25   26   27   28   29   30   31
32   33   34   35   36   37   38   39   40   41
42   43   44   45   46   47   48   49   50   51
52   53   54   55   56   57   58   59   60   61
62   63   64   65   66   67   68   69   70   71
72   73   74   75   76   77   78   79   80   81
82   83   84   85   86   87   88   89   90   91
92   93   94   95   96   97   98   99   100
101   102   103   104   105   106   107   108
109   110   111   112   113   114   115   116
117   118   119   120   121   122   123   124
125   126   127   128   129   130   131   132
133   134   135   136   137   138   139   140
141   142   143   144   145   146   147   148
149   150   151   152   153   154   155   156
157   158   159   160   161   162   163   164
165   166   167   168   169   170   171   172
173   174   175   176   177   178
```

```
Actualmente su juego de caracteres es:

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~

Ahora cambiamos al otro juego de
caracteres (MAYUSCULAS INVERTIDAS)
con VDP(4)=2

Ahora efectuamos 1000 cambios rapidos
entre los dos juegos de caracteres
Ok
```

Descripción de programa 4

Tenemos la posibilidad de visualizar en la pantalla sprites de 8*8 ó 16*16 puntos en su tamaño normal o ampliado.

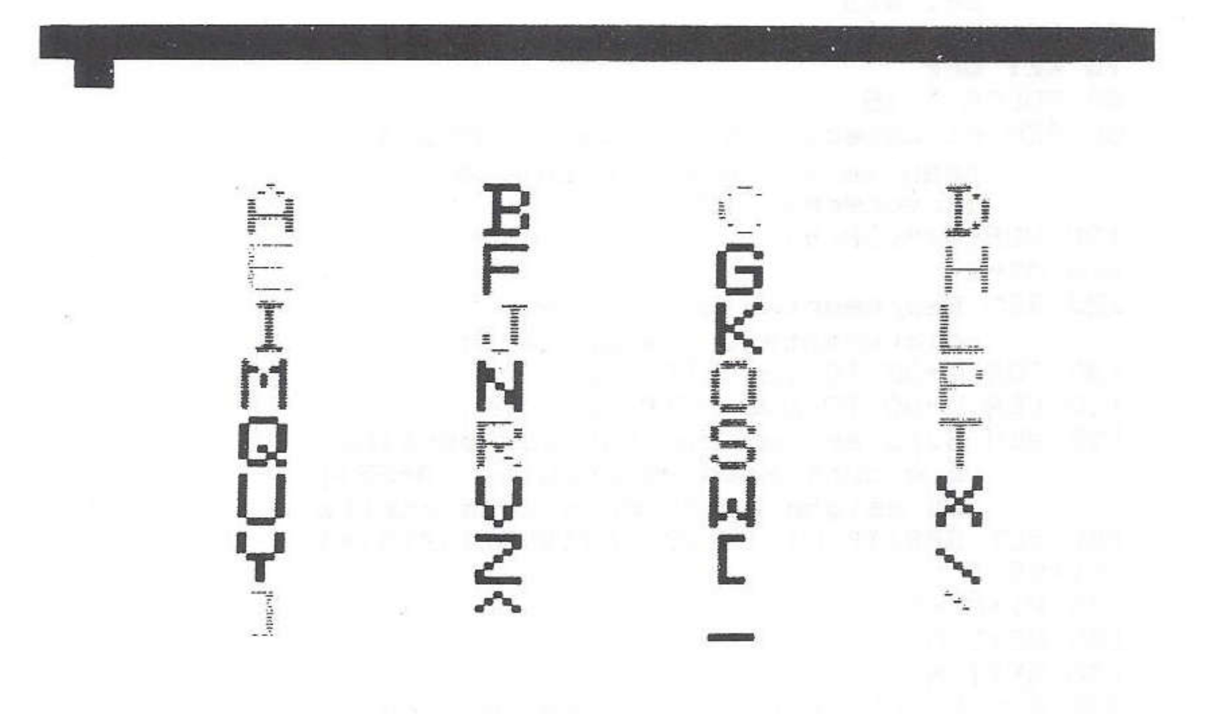
En la memoria VRAM pueden memorizarse hasta 256 sprites diferentes del tamaño 8*8, o bien 64 sprites del tamaño 16*16.

En la pantalla pueden visualizarse simultáneamente 32 de estos 256 ó 64 sprites, teniendo en cuenta que cada línea horizontal no admite más de 4 sprites al mismo tiempo. En caso contrario, el sprite de mayor número será borrado sucesiva o inmediatamente de la pantalla actual.

El programa 4 no sólo muestra la forma de poder visualizar 32 sprites en la pantalla sin provocar superposiciones horizontales de las imágenes. Su listado también le enseñará cómo cargar cualquier sprite a través del juego de caracteres estándar utilizando el comando VDP.

```
10 REM Programa 4
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                          Rainer Luers
40 REM Ocupacion de 32 sprites con las
       mayusculas (extraidas del juego
       de caracteres normal)
50 REM Sprites de 8*8 en representacion
       amliada
60 SCREEN 1,1
70 KEY OFF
80 COLOR 1,15
90 REM El aspecto de sprites (VDP(6))
       resulta del aspecto de los
       caracteres (VDP(4))
100 VDP(6)=VDP(4)
110 M1=0
120 REM Representar 32 sprites
        claramente en la pantalla
130 FOR N=32 TO 152 STEP 16
140 FOR M=50 TO 200 STEP 50
150 REM Solo se representan los sprites
        que contienen mayusculas (A=65);
        se asigna un color a cada sprite
160 PUT SPRITE M1,(M,N),INT(RND(1)*14)+1
,M1+65
170 M1=M1+1
180 NEXT M
190 NEXT N
200 REM Scroll pixel por pixel de los
        32 sprites al mismo tiempo
210 FOR L=1 TO 255
220 M1=0
230 FOR N=32 TO 152 STEP 16
240 FOR M=50 TO 200 STEP 50
```

```
250 PUT SPRITE M1,(M+L,N),,M1+65
260 M1=M1+1
270 NEXT M
280 NEXT N
290 NEXT L
```



Descripción de programa 5

Tenemos la posibilidad de visualizar en la pantalla sprites de 8*8 ó 16*16 puntos en su tamaño normal o ampliado.

En la memoria VRAM pueden memorizarse hasta 256 sprites diferentes del tamaño 8*8, ó bien 64 sprites del tamaño 16*16.

En la pantalla pueden visualizarse simultáneamente 32 de estos 256 o 64 sprites, teniendo en cuenta que cada línea horizontal no admite más de 4 sprites al mismo tiempo. En caso contrario, el sprite de mayor número será borrado sucesiva o inmediatamente de la pantalla actual.

El programa 5 demuestra ahora con varios ejemplos la forma en que se efectúa esta superposición de imágenes sucesiva, es decir, cuál de los sprites será el afectado por tal superposición en el caso en que se encuentren más de 4 sprites al mismo tiempo en una línea horizontal.

```
10 REM Programa 5
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                          Rainer Luers
40 REM Programa que sirve para
       demostrar que no pueden aparecer
       mas de 4 sprites en una
       horizontal al mismo tiempo; el
       sprite de mayor numero se va
       borrando sucesivamente
50 REM sprites de 8*8 con representacion
       ampliada
60 SCREEN 1,1
70 KEY OFF
80 COLOR 1,15
90 WIDTH 29
100 REM El aspecto de sprites (VDP(6))
        resulta del aspecto de los
        caracteres VDP(4))
110 VDP(6)=VDP(4)
120 REM Se comprueban tres estados
130 FOR L=1 TO 3
140 READ N0,N1,N2,N3,N4
150 REM Indicacion de los sprites fijos
160 PUT SPRITE N0,(25,96),,65
170 PUT SPRITE N1,(75,96),,66
180 PUT SPRITE N3,(150,96),,68
190 PUT SPRITE N4,(200,96),,69
200 LOCATE 1,10
210 REM Indicacion de los numeros de
        sprites
220 PRINT N0;TAB(7);N1;TAB(11);N2;TAB(16
);N3;TAB(22);N4
230 REM Se anade el quinto sprite
240 FOR N=70 TO 120
250 PUT SPRITE N2,(112,N),,67
260 REM Se emite el contenido de VDP(8)
```

```
270 A$=BIN$(VDP(8))
280 LOCATE 0,1
290 PRINT "VDP(8)=";A$
300 PRINT "mas de 4 sprites (1=si):";MID
$(A$,2,1)
310 PRINT "sprite cubierto:";:IF MID$(A$
,4,5)="11111" THEN PRINT "ninguno" ELSE
PRINT VAL("&b"+MID$(A$,4,5));"     "
320 REM Bucle de espera
330 FOR M=1 TO 200
340 NEXT M
350 NEXT N
360 NEXT L
370 LOCATE 0,0
380 END
390 REM DATAS de 3 estados
400 DATA 0,1,4,2,3
410 DATA 0,1,2,3,4
420 DATA 4,3,2,1,0
```

```
VDP(8)=11000100
mas de 4 sprites (1=si):1
sprite cubierto: 4


 0        1     4       2        3
A         B             D        E
                C
```

Descripción de programa 6

Este libro ha expuesto varias posibilidades de modificar el aspecto de caracteres en la pantalla.

Sin embargo, el problema se presenta después de la modificación de caracteres. Al cambiar de SCREEN o al desconectar el ordenador, éste olvida todos los cambios efectuados anteriormente y vuelve a su formato estándar.

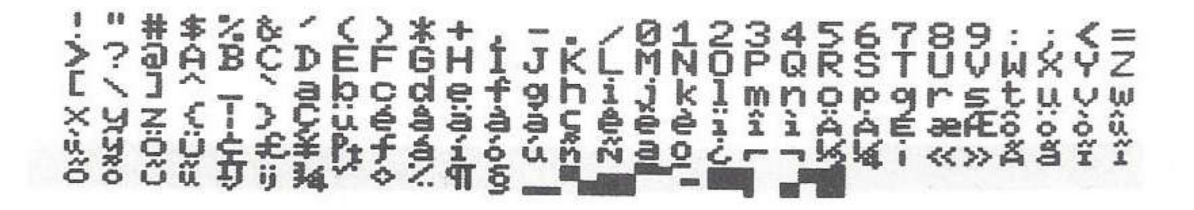
Por esta razón tendrá problemas si desea memorizar los juegos de caracteres modificados.

El programa 6 permite grabar cualquier juego de caracteres en cassette, suponiendo que la modificación haya sido efectuada en SCREEN 1.

No sólo eso: la grabación en cassette se realiza en forma de números de línea y líneas DATA correspondientes. De esta forma, con ayuda del comando MERGE puede cargar en cualquier momento el juego modificado para incluirlo de nuevo en otros programas.

```
Memorizar juego caracteres
--------------------------
Valor inicial ?  33
Valor final ?  200

Los caracteres escogidos por usted
tienen el aspecto siguiente:
```



```
Es esto correcto ( /N) ?

Nombre del juego ? MSX

Cassette listo grabar

Listo ( / ) ?
```

```
10 REM Programa 6
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                        Rainer Luers
40 REM Programa que almacena juegos de
       caracteres de la memoria (SCREEN
       1) en cassete, con la posiblidad
       de MERGEarlos mas tarde en forma
       de lineas DATA + rutina de carga
       en un programa existente
50 CLS
60 KEY OFF
70 REM SCREEN 1 debe estar activado
80 WIDTH 29
90 PRINT "Memorizar juego caracteres"
100 PRINT "--------------------------"
110 REM Limitar juego de caracteres
120 INPUT "Valor inicial ";M1
130 INPUT "Valor final ";M2
140 IF M1>M2 OR M2>255 OR M1<0 THEN RUN
150 PRINT
160 REM Indicacion de los caracteres
        escogidos
170 PRINT "Los caracteres escogidos
por usted tienen el         aspecto sig
uiente:"
180 PRINT
190 FOR N=M1 TO M2
200 PRINT CHR$(N);
210 NEXT N
220 PRINT
230 PRINT
240 INPUT "Es esto correcto ( /N) ";F$
250 IF F$<>"" THEN RUN
260 F$=""
270 PRINT
280 INPUT "Nombre del juego ";F$
```

```
290 PRINT
300 PRINT "Cassette listo grabar"
310 PRINT
320 INPUT "Listo ( / ) ";F1$
330 F$="cas:"+F$
340 OPEN F$ FOR OUTPUT AS #1
350 REM Activar linea inicial en 1000
360 Z=1000
370 REM Rutina de carga es confeccionada
        y grabada
380 A$=STR$(Z)+" FOR N ="+STR$(M1)+" TO"
+STR$(M2)
390 GOSUB 850
400 A$=STR$(Z)+" FOR M=0 TO 7"
410 GOSUB 850
420 A$=STR$(Z)+" READ A$"
430 GOSUB 850
440 A$=STR$(Z)+" VPOKE N*8+M"+CHR$(44)+"
VAL("+CHR$(34)+"&B"+CHR$(34)+"+A$)"
450 GOSUB 850
460 A$=STR$(Z)+" NEXT M"
470 GOSUB 850
480 A$=STR$(Z)+" NEXT N"
490 PRINT #1,A$
500 REM Grabar aspecto de caracteres
        en forma binaria
510 FOR N=M1 TO M2
520 REM Indicacion de los actuales
        caracteres y lineas de programa
530 LOCATE 16,CSRLIN-1
540 PRINT "Caracter";N
550 LOCATE 16,CSRLIN
560 PRINT "Linea ";
570 PRINT USING "####";Z+10;
580 Z=Z+10
590 REM Crear una linea de programa con
        numero de linea
```

```
600 A$=STR$(Z)+" REM  Caracter "+STR$(N)
610 PRINT #1,A$
620 FOR M=0 TO 7
630 A$=""
640 Z=Z+10
650 A$=STR$(Z)+" DATA "
660 REM Lectura del caracter en la
        memoria y conversion a
        representacion binaria
670 A1$=BIN$(VPEEK(N*8+M))
680 A1$=STRING$(8-LEN(A1$),"0")+A1$
690 A$=A$+A1$
700 REM Memorizacion de la linea Z
        completa
710 PRINT #1,A$
720 NEXT M
730 NEXT N
740 CLOSE
750 END
760 REM Rutina de test para comprobar
        el fichero almacenado en
        cassette
770 CLS
780 OPEN "cas:"FOR INPUT AS #1
790 LINE INPUT #1,A$
800 PRINT A$
810 REM Si no quedan Datas en cassette,
        cerrar el fichero
820 IF EOF(1)=-1 THEN CLOSE:END
830 GOTO 790
840 REM Grabar las lineas de programa y
        anadir 10 al numero de la
        linea Z
850 PRINT #1,A$
860 Z=Z+10
870 RETURN
```

Descripción de programa 7

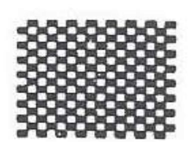
Tenemos la posibilidad de visualizar en la pantalla sprites de 8*8 ó 16*16 puntos en su tamaño normal o ampliado.

En la memoria VRAM pueden memorizarse hasta 256 sprites diferentes del tamaño 8*8, ó bien 64 sprites del tamaño 16*16.

En la pantalla pueden visualizarse simultáneamente 32 de estos 256 o 64 sprites, teniendo en cuenta que cada línea horizontal no admite más de 4 sprites al mismo tiempo. En caso contrario, el sprite de mayor número será borrado sucesiva o inmediatamente de la pantalla actual.

El programa 7 le ofrece la posibilidad de ir cambiando entre todos los tamaños de sprites arriba mencionados una vez definido el patrón de la figura móvil. Nuevamente el comando VDP le presta la ayuda necesaria para tal cometido: utilizándolo correctamente, todos los datos se mantienen en la VRAM, a diferencia de lo que sucede al utilizar el comando SCREEN.

Sprite 16*16
ampliado



```
10 REM Programa 7
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                          Rainer Luers
40 REM Cambiar entre representaciones de
       sprites de 8*8 y 16*16, cada uno
       con tamano normal y doble, sin
       efectuar comando SCREEN ni nueva
       definicion al cambiar
50 SCREEN 1
60 KEY OFF
70 COLOR 1,15
80 REM Lectura de sprites
       (16*16 puntos)
90 FOR N=1 TO 16
100 READ A$
110 REM Primero se lee el lado izquierdo
        (sprites 1 y 2 de 8*8)
120 A1$=A1$+CHR$(VAL("&B"+LEFT$(A$,8)))
130 NEXT N
140 RESTORE
150 FOR N=1 TO 16
160 READ A$
170 REM Aqui se lee el lado derecho de
        la representacion del sprite
        (sprites 3 y 4 de 8*8)
180 A1$=A1$+CHR$(VAL("&B"+RIGHT$(A$,8)))
190 NEXT N
200 REM Asignar el patron del sprite a
        la tabla de patrones de sprites
210 FOR N=14336 TO 14367
220 VPOKE N,ASC(MID$(A1$,N-14335,1))
230 NEXT N
240 REM Sprite gran tablero de ajedrez
250 DATA 0101010101010101
260 DATA 1010101010101010
270 DATA 0101010101010101
```

```
280 DATA 1010101010101010
290 DATA 0101010101010101
300 DATA 1010101010101010
310 DATA 0101010101010101
320 DATA 1010101010101010
330 DATA 0101010101010101
340 DATA 1010101010101010
350 DATA 0101010101010101
360 DATA 1010101010101010
370 DATA 0101010101010101
380 DATA 1010101010101010
390 DATA 0101010101010101
400 DATA 1010101010101010
410 CLS
420 LOCATE 0,0
430 PRINT "Sprite 8*8"
440 PRINT "no ampliado"
450 REM Bit 0 y bit 1 del registro 1
         son activados a 0, o sea,
         sprite 8*8 de tamano normal
460 VDP(1)=VDP(1) AND &HFC
470 GOSUB 740
480 CLS
490 LOCATE 0,0
500 PRINT "Sprite 8*8"
510 PRINT "ampliado"
520 REM Bit 0 del registro 1
         es activado a 1, o sea,
         sprite 8*8 de tamano doble
530 VDP(1)=VDP(1) OR 1
540 GOSUB 740
550 CLS
560 REM Bit 0 y bit 1 del registro 1
         son activados a 0, o sea,
         sprite 8*8 de tamano normal
570 VDP(1)=VDP(1) AND &HFC
580 LOCATE 0,0
```

```
590 PRINT "Sprite 16*16"
600 PRINT "no ampliado"
610 REM Bit 1 del registro 1
        es activado a 1, o sea,
        sprite 16*16 de tamano normal
620 VDP(1)=VDP(1) OR 2
630 GOSUB 740
640 CLS
650 LOCATE 0,0
660 PRINT "Sprite 16*16"
670 PRINT "ampliado"
680 REM Bit 0 del registro 1
        es activado a 1, o sea,
        sprite 16*16 de tamano doble
690 VDP(1)=VDP(1) OR 1
700 GOSUB 740
710 REM Se vuelve a ejecutar la demos-
        tracion sin necesidad de cargar
        de nuevo el patron del sprite
720 GOTO 410
730 REM Subprograma movimiento
        en diagonal del sprite
740 FOR N=1 TO 191
750 PUT SPRITE 0,(N,N),1,0
760 FOR M=1 TO 10
770 NEXT M
780 NEXT N
790 RETURN
```

Descripción de programa 8

El chip de sonido incorporado en el ordenador MSX permite crear sonidos independientes, así como emitirlos a través de tres canales de sonido al mismo tiempo.

Todos estos sonidos pueden crearse accediendo directamente a cada uno de los registros del chip, o bien utilizando el comando PLAY del BASIC MSX.

Sin embargo, como persona experimentada en tocar el piano, deberá orientarse por los nombres de las notas (utilizando PLAY) o de las diversas indicaciones numéricas (SOUND) para producir música; desgraciadamente el ordenador no dispone de un teclado de piano.

Para evitar este inconveniente he confeccionado el programa 8, que permite modificar la función del teclado de su ordenador de tal forma, que cada tecla produzca directamente un sonido diferente. En la pantalla se mostrarán las teclas que pueden ser pulsadas.

Paralelamente al sonido generado, se emitirá también la nota que ha sido tocada, de forma que además de la combinación 'tecla pulsada' y 'escuchar sonido' podrá estudiar también las notas.

Finalmente, las teclas de funciones 1 a 5 permiten generar acordes triples indicados previamente de forma directa a través del teclado (la entrada se efectuará tras previo borrado del campo de acordes mediante SHIFT junto con la correspondiente tecla de funciones; por ejemplo, borrar acorde 1 (F1) y entrarlo de nuevo con F6).

```
10 REM Programa 8
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                           Rainer Luers
40 REM Crear musica con teclado en una
       octava y media, indicacion de
       notas y llamada, es decir creador
       de acordes
50 REM Ejecucion, en lo que se refiere a
      la creacion de acordes, tras
      pulsar teclas de funcion
60 COLOR 15,4,4
70 ON KEY GOSUB 1280,1300,1320,1340,1360
,1390,1420,1450,1480,1510
80 FOR N=1 TO 10
90 KEY(N) ON
100 KEY N,""
110 NEXT N
120 REM Un sprite 8*8 en forma de flecha
          ampliada indica la ultima
          pulsacion de tecla
130 SCREEN 2,1
140 REM Dibujar las teclas blancas del
         piano
150 FOR N=24 TO 224 STEP 16
160 LINE (N,80)-(N+16,160),15,BF
170 LINE (N,80)-(N+16,160),1,B
180 NEXT N
190 REM Dibujar las teclas negras de.
         piano
200 FOR N=1 TO 9
210 READ A
220 LINE (A,80)-(A+16,128),1,BF
230 LINE (A,80)-(A+16,128),15,B
240 NEXT N
250 OPEN "grp:"FOR OUTPUT AS #1
260 COLOR 15
270 REM Dibujar los creadores de notas
         para las teclas blancas
```

```
280 FOR N=2 TO 14
290 READ A$
300 A=N*16
310 DRAW "bm =a;,168"
320 PRINT #1,A$
330 NEXT N
340 COLOR 1
350 REM Dibujar los creadores de notas
        para las teclas negras
360 FOR N=1 TO 9
370 READ A,A$
380 DRAW "bm =a;,64"
390 PRINT #1,A$
400 NEXT N
410 FOR N=1 TO 8
420 REM Lectura de la flecha-sprite
430 READ A$
440 A1$=A1$+CHR$(VAL("&b"+A$))
450 NEXT N
460 SPRITE$(1)=A1$
470 REM Dibujar las lineas de notas para
        notas aisladas y acordes
480 M1=24
490 GOSUB 1230
500 M1=64
510 GOSUB 1230
520 M1=96
530 GOSUB 1230
540 M1=128
550 GOSUB 1230
560 M1=160
570 GOSUB 1230
580 M1=192
590 GOSUB 1230
600 REM Comprobar pulsacion de tecla s/n
610 A$=INKEY$
620 IF A$="" GOTO 610
```

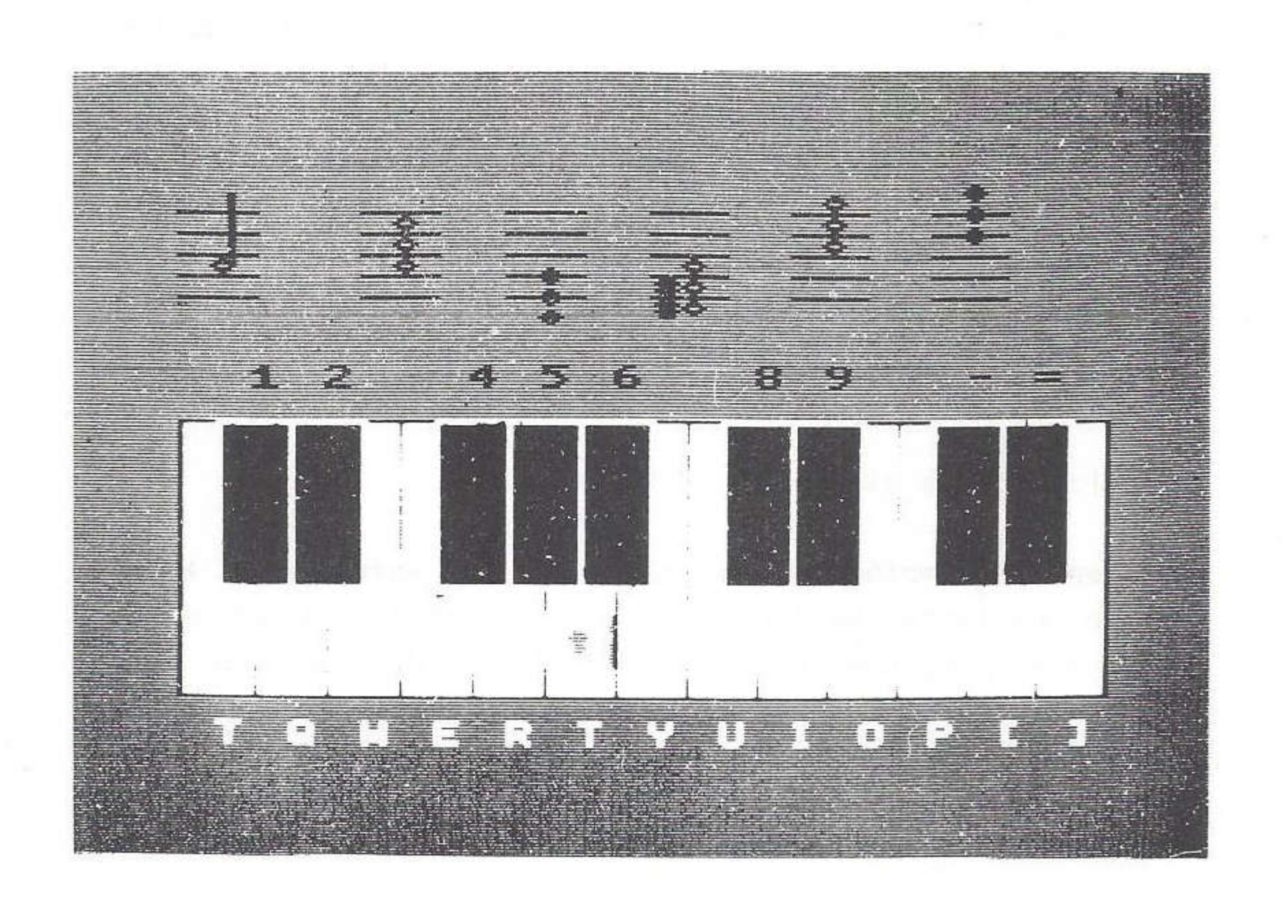
```
630 REM Teclas blancas
        Teclado 'TAB' - ']'
640 IF ASC(A$)=9 THEN PUT SPRITE 1,(24,1
35),4,1:A=50:B=4:C=0:N$="O4C":GOSUB 1030
650 IF ASC(A$)=113 THEN PUT SPRITE 1,(40
,135),4,1:A=47:B=0:C=0:N$="O4D":GOSUB 10
30
660 IF ASC(A$)=119 THEN PUT SPRITE 1,(56
,135),4,1:A=44:B=0:C=0:N$="O4E":GOSUB 10
30
670 IF ASC(A$)=101 THEN PUT SPRITE 1,(72
,135),4,1:A=41:B=0:C=0:N$="O4F":GOSUB 10
30
680 IF ASC(A$)=114 THEN PUT SPRITE 1,(88
,135),4,1:A=38:B=0:C=0:N$="O4G":GOSUB 10
30
690 IF ASC(A$)=116 THEN PUT SPRITE 1,(10
4,135),4,1:A=35:B=0:C=0:N$="O4A":GOSUB 1
030
700 IF ASC(A$)=121 THEN PUT SPRITE 1,(12
0,135),4,1:A=32:B=0:C=0:N$="O4B":GOSUB 1
130
710 IF ASC(A$)=117 THEN PUT SPRITE 1,(13
6,135),4,1:A=29:B=0:C=0:N$="O5C":GOSUB 1
130
720 IF ASC(A$)=105 THEN PUT SPRITE 1,(15
2,135),4,1:A=26:B=0:C=0:N$="O5D":GOSUB 1
130
730 IF ASC(A$)=111 THEN PUT SPRITE 1,(16
8,135),4,1:A=23:B=0:C=0:N$="O5E":GOSUB 1
130
740 IF ASC(A$)=112 THEN PUT SPRITE 1,(18
4,135),4,1:A=20:B=0:C=0:N$="O5F":GOSUB 1
130
750 IF ASC(A$)=91 THEN PUT SPRITE 1,(200
,135),4,1:A=17:B=0:C=0:N$="O5G":GOSUB 11
30
```

```
760 IF ASC(A$)=93 THEN PUT SPRITE 1,(216
,135),4,1:A=14:B=4:C=0:N$="O5A":GOSUB 11
30
770 REM Teclas negras
        Teclado '1' - '='
780 IF ASC(A$)=49 THEN PUT SPRITE 1,(32,
100),4,1:A=50:B=4:C=47:N$="O4C#":GOSUB 1
030
790 IF ASC(A$)=50 THEN PUT SPRITE 1,(48,
100),4,1:A=47:B=0:C=44:N$="O4D#":GOSUB 1
030
800 IF ASC(A$)=52 THEN PUT SPRITE 1,(80,
100),4,1:A=41:B=0:C=38:N$="O4F#":GOSUB 1
030
810 IF ASC(A$)=53 THEN PUT SPRITE 1,(96,
100),4,1:A=38:B=0:C=35:N$="O4G#":GOSUB 1
030
820 IF ASC(A$)=54 THEN PUT SPRITE 1,(112
,100),4,1:A=35:B=0:C=32:N$="O4A#":GOSUB
1030
830 IF ASC(A$)=56 THEN PUT SPRITE 1,(144
,100),4,1:A=29:B=0:C=26:N$="O5C#":GOSUB
1130
840 IF ASC(A$)=57 THEN PUT SPRITE 1,(160
,100),4,1:A=26:B=0:C=23:N$="O5D#":GOSUB
1130
850 IF ASC(A$)=45 THEN PUT SPRITE 1,(192
,100),4,1:A=20:B=0:C=17:N$="O5F#":GOSUB
1130
860 IF ASC(A$)=61 THEN PUT SPRITE 1,(208
,100),4,1:A=17:B=0:C=14:N$="O5G#":GOSUB
1130
870 GOTO 610
880 REM DATAs de indicacion en pantalla
       del teclado
890 DATA 32,48,80,96,112,144,160,192,208
900 DATA T,Q,W,E,R,T,Y,U,I,O,P,[,]
```

```
910 DATA 40,1,56,2,88,4,104,5,120,6,152,
8,168,9,200,-,216,=
920 REM Patron de sprite 8*8 para flecha
930 DATA 11111111
940 DATA 10000001
950 DATA 10011001
960 DATA 10111101
970 DATA 10011001
980 DATA 10011001
990 DATA 10000001
1000 DATA 11111111
1010 GOTO 1010
1020 REM Entrada acorde (AK) s/n; si no,
        la entrada e indicacion solo
        vale para una nota con el palo
        hacia arriba
1030 IF AK>0 THEN GOTO 1550
1040 LINE(24,10)-(40,54),4,BF
1050 PLAY N$
1060 M1=24
1070 GOSUB 1230
1080 DRAW "bm 35,=a;h2g2f2e2nl=b;u20"
1090 REM Tecla negra significa: '#'
1100 IF C<>0 THEN DRAW "bm25,=c;":PRINT
#1,"#"
1110 RETURN
1120 REM Entrada acorde (AK) s/n; si no,
        la entrada e indicacion solo
        vale para una nota con el palo
        hacia abajo
1130 IF AK>0 THEN GOTO 1550
1140 LINE(24,10)-(40,54),4,BF
1150 PLAY N$
1160 M1=24
1170 GOSUB 1230
1180 DRAW "bm 35,=a;e2f2g2h2nr=b;d20"
1190 REM Tecla negra significa: '#'
```

```
1200 IF C<>0 THEN DRAW "bm28,-c;":PRINT
#1,"#"
1210 RETURN
1220 REM Dibujar las lineas de notas
1230 FOR N=20 TO 44 STEP 6
1240 LINE (M1,N)-(M1+16,N),1
1250 NEXT N
1260 RETURN
1270 REM Subprogramas de F1 y F5
         suena el acorde
1280 PLAY A$(1),A$(2),A$(3)
1290 RETURN
1300 PLAY B$(1),B$(2),B$(3)
1310 RETURN
1320 PLAY C$(1),C$(2),C$(3)
1330 RETURN
1340 PLAY D$(1),D$(2),D$(3)
1350 RETURN
1360 PLAY E$(1),E$(2),E$(3)
1370 RETURN
1380 REM Lectura de los acordes 1 a 5,
         con previo borrado de los
         ultimos acordes
1390 AK=75:M1=64
1400 GOSUB 1680:GOSUB 1230
1410 RETURN
1420 AK=107:M1=96
1430 GOSUB 1680:GOSUB 1230
1440 RETURN
1450 AK=139:M1=128
1460 GOSUB 1680:GOSUB 1230
1470 RETURN
1480 AK=171:M1=160
1490 GOSUB 1680:GOSUB 1230
1500 RETURN
1510 AK=203:M1=192
1520 GOSUB 1680:GOSUB 1230
```

```
1530 RETURN
1540 REM Las tres notas de cada acorde
          son emitidas en pantalla y
          tocadas
1550 ZZ=ZZ+1
1560 PLAY N$
1570 DRAW "bm=ak;,=a;c1h2g2f2e21=b;"
1580 REM Tecla negra significa: '#'
1590 IF C<>0 THEN TT=AK-10:DRAW "bm=tt;,
=c;":PRINT #1,"#"
1600 IF AK=75 THEN A$(ZZ)=N$
1610 IF AK=107 THEN B$(ZZ)=N$
1620 IF AK=139 THEN C$(ZZ)=N$
1630 IF AK=171 THEN D$(ZZ)=N$
1640 IF AK=203 THEN E$(ZZ)=N$
1650 IF ZZ=3 THEN ZZ=0:AK=0
1660 RETURN
1670 REM Borrado del acorde anterior
1680 LINE(AK-11,10)-(AK-11+16,54),4,BF
1690 RETURN
```



Descripción de programa 9

El chip de sonido incorporado en el ordenador MSX permite crear sonidos independientes, así como emitirlos a través de tres canales de sonido al mismo tiempo.

Todos estos sonidos pueden crearse accediendo directamente a cada uno de los registros del chip, o bien utilizando el comando PLAY del BASIC MSX.

Sin embargo, no es tan fácil generar sonidos un poco más complicados a través del comando PLAY, sin haber hojeado previamente el manual de referencia,para saber cómo utilizar los diversos parámetros del comando PLAY.

El programa 9 no le evita esta tarea: indicando la correspondiente letra (por ejemplo, 'o' de octava o 'v' de volumen), puede facilitarse la programación mediante PLAY, de forma que ahora modifique el valor numérico utilizando las teclas del cursor hacia izquierda y derecha. Mientras tanto, el programa evita automáticamente la entrada de valores ilegales (p.e. octava mayor que '8').

Si desea conectar el sintetizador (modificación de la curva envolvente a través del parámetro 's' de PLAY), deberá activarlo previamente pulsando las teclas CTRL y 's' (esta combinación servirá igualmente para volver a desconectarlo).

La música sonará al pulsar la tecla ENTER.

Finalmente, también puede almacenar sus entradas de forma que se conviertan en líneas DATA,para permitir su posterior carga a un programa BASIC mediante MERGE. Para activar esta grabación, deberá indicar un nombre de programa al principio y pulsar la tecla ESC para grabar cada sonido. La combinación de las teclas CTRL-STOP finalizará el programa y la grabación de los datos.

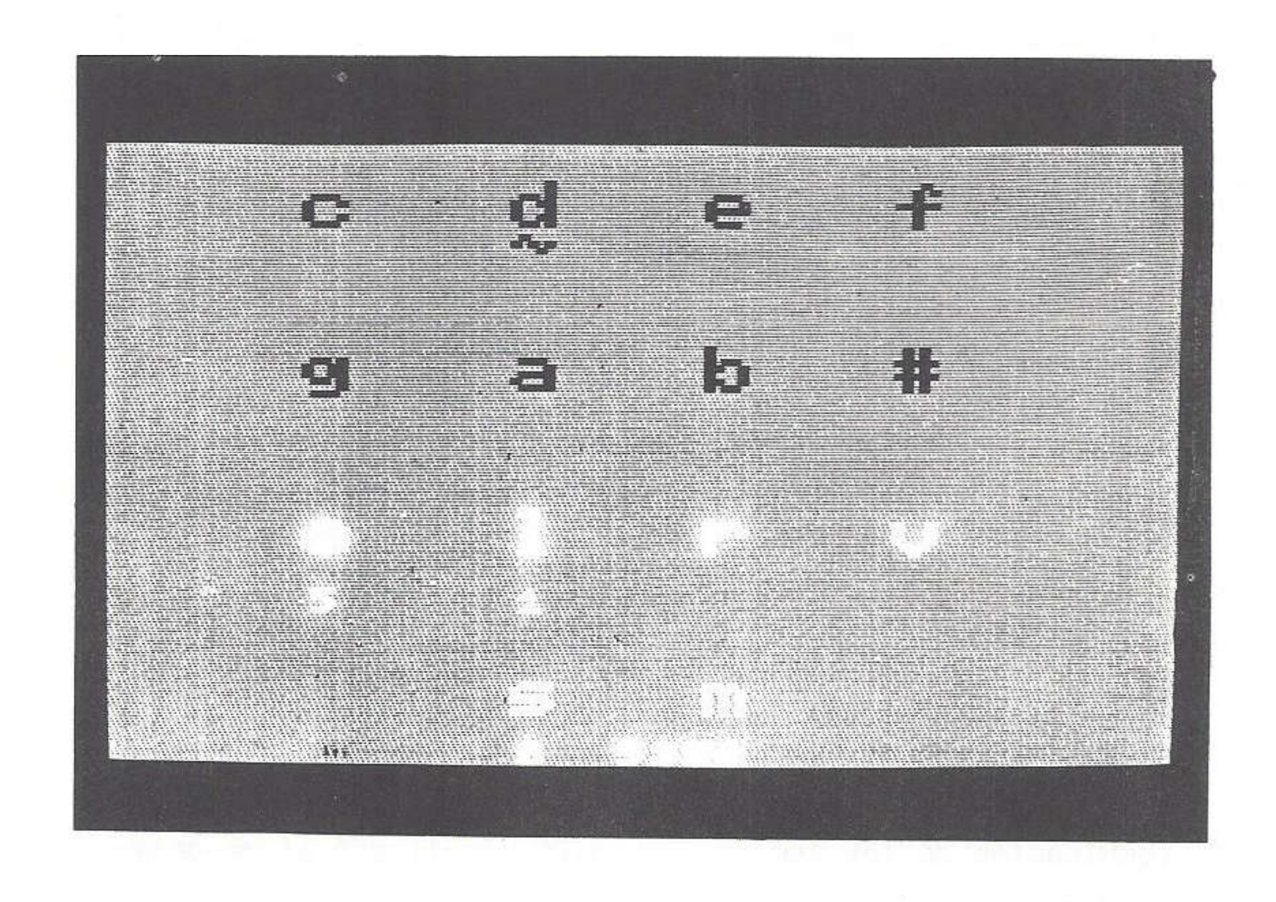
```
10 REM Programa 9
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                          Rainer Luers
40 REM Programa, que no solo sirve para
       probar sonidos con el editor de
       comando PLAY, sino ademas para
       volver a utilizarlos mas tarde
       en otros programas
50 REM Proteccion de interrupcion, para
       poder cerrar ficheros DATA
60 ON STOP GOSUB 1100
70 STOP ON
80 SCREEN 1,1
90 REM Grabacion de strings
       PLAY s/n?
100 INPUT "PLAY en DATA (S/ ) ";F$
110 IF F$<>"" THEN INPUT "Nombre ";F$:TT
=990:F$="cas:"+F$:OPEN F$ FOR OUTPUT AS
#1
120 CLS
130 KEY OFF
140 COLOR 15,4,1
150 REM Sprites 8*8 se cargan con el
        juego de caracteres normal
160 VDP(6)=VDP(4)
170 REM Valores iniciales de los
        diversos parametros PLAY
180 O=4
190 L=4
200 R=64
210 V=8
220 S=13
230 M=200
240 REM Curva envolvente activada   :SS=1
        Curva envolvente desactiv. :SS=0
250 SS=0
```

```
260 REM Dibujar los nombres de las notas
        en la pantalla
270 PUT SPRITE 0,(46,10),1,99
280 PUT SPRITE 1,(94,10),1,100
290 PUT SPRITE 2,(140,10),1,101
300 PUT SPRITE 3,(186,10),1,102
310 PUT SPRITE 4,(46,60),1,103
320 PUT SPRITE 5,(94,60),1,97
330 PUT SPRITE 6,(140,60),1,98
340 PUT SPRITE 7,(186,60),1,35
350 REM Dibujar o(ctava)
                l(ongitud)
                r(esto=pausa)
                v(olumen)
360 PUT SPRITE 8,(46,110),15,111
370 PUT SPRITE 9,(94,110),15,108
380 PUT SPRITE 10,(140,110),15,114
390 PUT SPRITE 11,(186,110),15,118
400 REM Dibujar s=curva envolvente
                m=sinton.fina de c.e.
410 PUT SPRITE 12,(94,160),10,115
420 PUT SPRITE 13,(140,160),10,109
430 REM Lectura de entrada
440 A$=INKEY$
450 IF A$="" THEN GOTO 440
460 REM Que debe hacerse cuando una nota
        ha sido activada?
470 IF A$="c" THEN PUT SPRITE 14,(46,26)
,1,126:N$="c":GOTO 440
480 IF A$="d" THEN PUT SPRITE 14,(94,26)
,1,126:N$="d":GOTO 440
490 IF A$="e" THEN PUT SPRITE 14,(140,26
),1,126:N$="e":GOTO 440
500 IF A$="f" THEN PUT SPRITE 14,(186,26
),1,126:N$="f":GOTO 440
510 IF A$="g" THEN PUT SPRITE 14,(46,76)
,1,126:N$="g":GOTO 440
```

```
520 IF A$="a" THEN PUT SPRITE 14,(94,76)
,1,126:N$="a":GOTO 440
530 IF A$="b" THEN PUT SPRITE 14,(140,76
),1,126:N$="b":GOTO 440
540 IF A$="#" THEN PUT SPRITE 14,(186,76
),1,126:N$=N$+"#":GOTO 440
550 REM Que debe hacerse, cuando se ha
        activado o=Oktave?
560 IF A$="o" THEN PUT SPRITE 14,(46,126
),15,126 ELSE GOTO 630
570 IF O=8 THEN O=0
580 O=O+1
590 LOCATE 4,17
600 PRINT USING "#";O;
610 GOTO 440
620 REM Que debe hacerse cuando se ha
        activado l=longitud?
630 IF A$="l" THEN PUT SPRITE 14,(94,126
),15,126 ELSE GOTO 700
640 IF L=64 THEN L=0
650 L=L+1
660 LOCATE 9,17
670 PRINT USING "##";L;
680 GOTO 440
690 REM Que debe hacerse cuando se ha
        activado r=pausa?
700 IF A$="r" THEN PUT SPRITE 14,(140,12
6),15,126 ELSE GOTO 770
710 IF R=64 THEN R=0
720 R=R+1
730 LOCATE 15,17
740 PRINT USING "##";R;
750 GOTO 440
760 REM Que debe hacerse cuando se ha
        activado v=volumen?
770 IF A$="v" THEN PUT SPRITE 14,(186,12
6),15,126 ELSE GOTO 840
```

```
780 IF V=15 THEN V=-1
790 V=V+1
800 LOCATE 21,17
810 PRINT USING "##";V;
820 GOTO 440
830 REM Que debe hacerse cuando se ha
        activado s=curva envolvente?
840 IF A$="s" THEN PUT SPRITE 14,(94,176
),10,126 ELSE GOTO 910
850 IF S=14 THEN S=0
860 S=S+1
870 LOCATE 9,23
880 PRINT USING "##";S;
890 GOTO 440
900 REM Que debe hacerse cuando se ha
        activado m=sintonizacion fina
        de curva envolvente?
910 IF A$="m" THEN PUT SPRITE 14,(140,17
6),10,126 ELSE GOTO 960
920 IF M=65500! THEN M=0
930 M=M+100
940 LOCATE 12,23
950 PRINT USING "#####";M;
960 IF A$="M" THEN PUT SPRITE 14,(140,17
6),10,126 ELSE GOTO 1030
970 IF M=100 THEN M=65600!
980 M=M-100
990 LOCATE 12,23
1000 PRINT USING "#####";M;
1010 GOTO 440
1020 REM Que debe hacerse cuando se ha
        activado CTRL S=interruptor de
        la curva envolvente?
1030 IF ASC(A$)=19 THEN IF SS=0 THEN SS=
1 ELSE SS=0:GOTO 440
1040 REM Que debe hacerse cuando se ha
        activado ENTER=emitir un
        sonido?
```

```
1050 IF ASC(A$)=13 THEN IF SS=1 THEN PLA
Y "o=o;l=l;r=r;v=v;s=s;m=m;xn$;" ELSE PL
AY "o=o;l=l;r=r;v=v;xn$;"
1060 REM Que debe hacerse cuando se ha
         activado ESC=grabar el estado
         de PLAY?
1070 IF ASC(A$)=27 AND TT>0 THEN TT=TT+1
0:A1$=STR$(TT)+" DATA "+"o"+STR$(O)+"l"+
STR$(L)+"r"+STR$(R):IF SS=0 THEN A1$=A1$
+"v"+STR$(V)+N$:PRINT #1,A1$ ELSE A1$=A1
$+"s"+STR$(S)+"m"+STR$(M)+N$:PRINT #1,A1
$
1080 GOTO 440
1090 REM Subprograma para finalizar el
         programa con CTRL-STOP
1100 CLOSE
1110 SCREEN 0
1120 END
```



Descripción de programa 10

Los poseedores del libro 'Programas y utilidades del MSX' de DATA BECKER ya conocerán gran parte del programa 10. Analicemos cada una de sus características:

El chip de sonido incorporado en el ordenador MSX permite crear sonidos independientes, así como emitirlos a través de tres canales de sonido al mismo tiempo.

Todos estos sonidos pueden crearse accediendo directamente a cada uno de los registros del chip, o bien utilizando el comando PLAY del BASIC MSX.

Sin embargo, no es tan fácil producir sonido mediante el comando SOUND, puesto que en este caso no basta con acceder simplemente a un canal de música; también deben activarse los registros de volumen y de combinaciones con los valores adecuados para generar finalmente sonidos. Además, para crear un timbre desgarrado, deberá activar previamente a '16' el volumen al menos de un canal de sonido.

Seguramente ya habrá probado infinidad de cosas, y probablemente haya abandonado los intentos frustrados.

El programa 10 no sólo permite acceder a los registros del chip de sonido a través de las teclas del cursor (flechas arriba y abajo), sino también modificar los valores de dichos registros (flechas izquierda y derecha). Al pulsar la barra espaciadora dará la orden de emitir el sonido.

Finalmente, también puede almacenar sus entradas de forma que se conviertan en líneas DATA, para permitir su posterior carga a un programa BASIC mediante MERGE. Para activar esta grabación, deberá indicar un nombre de programa al principio y pulsar la tecla ESC para grabar cada sonido. La combinación de las teclas CTRL-STOP finalizará el programa y la grabación de los datos.

```
10 REM Programa 10
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                      Rainer Luers
40 REM Este editor de sonidos no solo le
       permite entrar facilmente los so-
       nidos que ha creado (pulse las
       teclas del cursor), sino tambien
       escucharlos (barra espaciadora)
50 REM y, si lo desea, grabarlos en
       lineas DATA para anadirlas con
       MERGE a programas posteriores
60 REM Tomar precauciones
70 REM Proteger contra interrupcion,
       para cerrar ficheros DATA
80 SCREEN 0:WIDTH 35
90 CLEAR 2000:DIM A$(21),A(21)
100 ON STOP GOSUB 1280
110 STOP ON
120 KEY OFF
130 COLOR 15,1,1
140 REM Grabacion de los
        parametros SOUND s/n?
150 INPUT "SOUND en DATA (S/ ) ";F$
160 IF F$<>"" THEN INPUT "Nombre ";F$:TT
=990:F$="cas:"+F$:OPEN F$ FOR OUTPUT AS
#1
170 CLS
180 REM Devolver todos los parametros
        de sonido a su estado original
190 FOR N=0 TO 13
200 SOUND N,0
210 NEXT N
220 SOUND 7,63
230 REM Configuracion de la pantalla
240 PRINT TAB(8) "Editor de sonido"
```

```
250 PRINT TAB(8) "================"
260 PRINT
270 PRINT TAB(5);:A$(3)="Ajuste fino can
al 1:":PRINT A$(3)
280 PRINT TAB(3);:A$(4)="Ajuste aprox. c
anal 1:":PRINT A$(4)
290 PRINT TAB(5);:A$(5)="Ajuste fino can
al 2:":PRINT A$(5)
300 PRINT TAB(3);:A$(6)="Ajuste aprox. c
anal 2:":PRINT A$(6)
310 PRINT TAB(5);:A$(7)="Ajuste fino can
al 3:":PRINT A$(7)
320 PRINT TAB(3);:A$(8)="Ajuste aprox. c
anal 3:":PRINT A$(8)
330 PRINT
340 PRINT TAB(6);:A$(10)="Canal 1: 0/1=T
/2=R:":PRINT A$(10)
350 PRINT TAB(6);:A$(11)="Canal 2: 0/1=T
/2=R:":PRINT A$(11)
360 PRINT TAB(6);:A$(12)="Canal 3: 0/1=T
/2=R:":PRINT A$(12)
370 PRINT
380 PRINT TAB(9);:A$(14)="Volumen canal
1:":PRINT A$(14)
390 PRINT TAB(9);:A$(15)="Volumen canal
2:":PRINT A$(15)
400 PRINT TAB(9);:A$(16)="Volumen canal
3:":PRINT A$(16)
410 PRINT
420 PRINT TAB(9);:A$(18)="Periodo zumbid
o:":PRINT A$(18)
430 PRINT TAB(2);:A$(19)="Periodo finu c
.envolv.:":PRINT A$(19)
440 PRINT TAB(3);:A$(20)="Periodo aprox.
 c.env.:":PRINT A$(20)
450 PRINT TAB(8);:A$(21)="Curva envolven
te:":PRINT A$(21)
```

```
460 IF TT<>0 THEN PRINT:PRINT "Grabacion
 en lineas DATA";
470 REM Asignar valores a parametros
       de sonido en la pantalla
480 FOR N=3 TO 8
490 LOCATE 26,N
500 PRINT A
510 NEXT N
520 FOR N=10 TO 12
530 LOCATE 26,N
540 PRINT A
550 NEXT N
560 FOR N=14 TO 16
570 LOCATE 26,N
580 PRINT A
590 NEXT N
600 FOR N=18 TO 21
610 LOCATE 26,N
620 PRINT A
630 NEXT N
640 REM Colocar el cursor de lineas en
       la posicion 0 de la correspon-
       diente linea de pantalla
650 LOCATE 0,3:PRINT CHR$(200);
660 REM Esperar entrada; las posibilida-
       des de entrada se refieren solo
       al campo de control del cursor,
       o sea flecha hacia arriba o
            flecha hacia abajo ->
            posicionamiento cursor
670 REM     flecha hacia izquierda o
            flecha hacia derecha ->
            parametro de sonido +/-
            barra espaciadora -> SOUND
            ESC -> grabacion
680 A$=INKEY$:IF A$="" THEN GOTO 680
690 REM Que debe hacerse cuando se ha
       activado ESC=grabar los
       parametros de SOUND?
```

```
700 IF ASC(A$)=27 AND TT>0 THEN TT=TT+10
:A1$=STR$(TT)+" DATA "+STR$(A(3))+","+ST
R$(A(4))+","+STR$(A(5))+","+STR$(A(6))+"
,"+STR$(A(7))+","+STR$(A(8))+","+STR$(A(
18))+","+STR$(A) ELSE GOTO 730
710 A1$=A1$+","+STR$(A(14))+","+STR$(A(1
5))+","+STR$(A(16))+","+STR$(A(19))+","+
STR$(A(20))+","+STR$(A(21))
720 PRINT #1,A1$
730 IF A$=" " THEN SOUND 13,A(21):GOTO 6
80
740 IF ASC(A$)=30 OR ASC(A$)=31 THEN GOT
O 750 ELSE IF ASC(A$)=28 OR ASC(A$)=29 T
HEN GOTO 830 ELSE GOTO 680
750 IF ASC(A$)=30 AND CSRLIN=3 THEN GOTO
 680
760 IF ASC(A$)=31 AND CSRLIN=21 THEN GOT
O 680
770 IF ASC(A$)=31 THEN IF CSRLIN=8 OR CS
RLIN=12 OR CSRLIN=16 THEN A=2 ELSE A=1 E
LSE GOTO 800
780 LOCATE 0,CSRLIN:PRINT " ";:LOCATE 0,
CSRLIN+A:PRINT CHR$(200);:LOCATE 25,CSRL
IN
790 GOTO 680
800 IF ASC(A$)=30 THEN IF CSRLIN=18 OR C
SRLIN=14 OR CSRLIN=10 THEN A=-2 ELSE A=-
1 ELSE GOTO 680
810 LOCATE 0,CSRLIN:PRINT " ";:LOCATE 0,
CSRLIN+A:PRINT CHR$(200);:LOCATE 25,CSRL
IN
820 GOTO 680
830 IF ASC(A$)=28 THEN Z=1 ELSE Z=-1
840 LOCATE 26,CSRLIN:PRINT "   ";:LOCATE
 26,CSRLIN
```

```
850 REM Entrar parametros de sonido
        + oder - al comando 'SOUND';
        comprobar ademas llegada a los
        valores limite, asi como contro-
        lar indicacion en pantalla
860 REM Ajuste tono fino canal 1
        ========================
870 IF CSRLIN=3 THEN IF (A(3)+Z<0 OR A(3
)+Z>255) THEN PRINT A(3);:GOTO 680 ELSE
A(3)=A(3)+Z:PRINT A(3);:SOUND 0,A(3):GOT
O 680
880 REM Ajuste tono aprox canal 1
        =========================
890 IF CSRLIN=4 THEN IF (A(4)+Z<0 OR A(4
)+Z>15) THEN PRINT A(4);:GOTO 680 ELSE A
(4)=A(4)+Z:PRINT A(4);:SOUND 1,A(4):GOTO
 680
900 REM Ajuste tono fino canal 2
        ========================
910 IF CSRLIN=5 THEN IF (A(5)+Z<0 OR A(5
)+Z>255) THEN PRINT A(5);:GOTO 680 ELSE
A(5)=A(5)+Z:PRINT A(5);:SOUND 2,A(5):GOT
O 680
920 REM Ajuste tono aprox canal 2
        =========================
930 IF CSRLIN=6 THEN IF (A(6)+Z<0 OR A(6
)+Z>15) THEN PRINT A(6);:GOTO 680 ELSE A
(6)=A(6)+Z:PRINT A(6);:SOUND 3,A(6):GOTO
 680
940 REM Ajuste tono fino canal 3
        ========================
950 IF CSRLIN=7 THEN IF (A(7)+Z<0 OR A(7
)+Z>255) THEN PRINT A(7);:GOTO 680 ELSE
A(7)=A(7)+Z:PRINT A(7);:SOUND 4,A(7):GOT
O 680
960 REM Ajuste tono aprox canal 3
        =========================
```

```
970 IF CSRLIN=8 THEN IF (A(8)+Z<0 OR A(8
)+Z>15) THEN PRINT A(8);:GOTO 680 ELSE A
(8)=A(8)+Z:PRINT A(8);:SOUND 5,A(8):GOTO
 680
980 REM Volumen canal 1
        ===============
990 IF CSRLIN=14 THEN IF (A(14)+Z<0 OR A
(14)+Z>16) THEN PRINT A(14);:GOTO 680 EL
SE A(14)=A(14)+Z:PRINT A(14);:SOUND 8,A(
14):GOTO 680
1000 REM Volumen canal 2
         ===============
1010 IF CSRLIN=15 THEN IF (A(15)+Z<0 OR
A(15)+Z>16) THEN PRINT A(15);:GOTO 680 E
LSE A(15)=A(15)+Z:PRINT A(15);:SOUND 9,A
(15):GOTO 680
1020 REM Volumen canal 3
         ===============
1030 IF CSRLIN=16 THEN IF (A(16)+Z<0 OR
A(16)+Z>16) THEN PRINT A(16);:GOTO 680 E
LSE A(16)=A(16)+Z:PRINT A(16);:SOUND 10,
A(16):GOTO 680
1040 REM Periodo zumbido
         ===============
1050 IF CSRLIN=18 THEN IF (A(18)+Z<0 OR
A(18)+Z>31) THEN PRINT A(18);:GOTO 680 E
LSE A(18)=A(18)+Z:PRINT A(18);:SOUND 6,A
(18):GOTO 680
1060 REM Periodo fino curva envolvente
         ============================
1070 IF CSRLIN=19 THEN IF (A(19)+Z<0 OR
A(19)+Z>255) THEN PRINT A(19);:GOTO 680
ELSE A(19)=A(19)+Z:PRINT A(19);:SOUND 11
,A(19):GOTO 680
1080 REM Periodo aprox. curva envolvente
         ==============================
```

```
1090 IF CSRLIN=20 THEN IF (A(20)+Z<0 OR
A(20)+Z>255) THEN PRINT A(20);:GOTO 680
ELSE A(20)=A(20)+Z:PRINT A(20);:SOUND 12
,A(20):GOTO 680
1100 REM Curva envolvente
         ================
1110 IF CSRLIN=21 THEN IF (A(21)+Z<0 OR
A(21)+Z>15) THEN PRINT A(21);:GOTO 680 E
LSE A(21)=A(21)+Z:PRINT A(21);:SOUND 13,
A(21):GOTO 680
1120 REM Canal 1: apagado/ton/zum/ambos
         ===============================
1130 IF CSRLIN=10 THEN IF (A(10)+Z<0 OR
A(10)+Z>3) THEN PRINT A(10);:GOTO 680 EL
SE A(10)=A(10)+Z:PRINT A(10);
1140 REM Canal 2: apagado/ton/zum/ambos
         ===============================
1150 IF CSRLIN=11 THEN IF (A(11)+Z<0 OR
A(11)+Z>3) THEN PRINT A(11);:GOTO 680 EL
SE A(11)=A(11)+Z:PRINT A(11);
1160 REM Canal 3: apagado/ton/zum/ambos
         ===============================
1170 IF CSRLIN=12 THEN IF (A(12)+Z<0 OR
A(12)+Z>3) THEN PRINT A(12);:GOTO 680 EL
SE A(12)=A(12)+Z:PRINT A(12);
1180 IF CSRLIN<10 OR CSRLIN>12 THEN GOTO
 1260
1190 REM Calculo exacto del parametro 7
         de 'SOUND'
1200 A=0
1210 IF A(10)=1 THEN A=A+8 ELSE IF A(10)
=2 THEN A=A+1 ELSE IF A(10)=0 THEN A=A+1
+8
1220 IF A(11)=1 THEN A=A+16 ELSE IF A(11
)=2 THEN A=A+2 ELSE  IF A(11)=0 THEN A=A
+16+2
```

```
1230 IF A(12)=1 THEN A=A+32 ELSE IF A(12
)=2 THEN A=A+4 ELSE IF A(12)=0 THEN A=A+
32+4
1240 A=A+128
1250 SOUND 7,A
1260 GOTO 680
1270 WIDTH 40
1280 REM Subprograma para finalizar el
        programa con CTRL STOP
1290 CLOSE
1300 SCREEN 0
1310 END
```

```
            Editor de sonido
            ================

     Ajuste  fino   canal 1:   107
     Ajuste aprox.  canal 1:   13
     Ajuste  fino   canal 2:   2
     Ajuste aprox.  canal 2:   9
     Ajuste  fino   canal 3:   150
     Ajuste aprox.  canal 3:   6

      Canal 1: O/1=T/2=R  :   1
      Canal 2: O/1=T/2=R  :   2
      Canal 3: O/1=T/2=R  :   1

     Volumen  canal 1     :   16
     Volumen  canal 2     :   7
     Volumen  canal 3     :   12

         Periodo zumbido  :   2
  Periodo fino c.envolv.  :   19
  Periodo aprox. c.env.   :   1
       Curva envolvente   :   8

 Grabacion en lineas DATA
```

Descripción de programa 11

Como puede ver en este libro, su ordenador MSX ofrece una gran variedad de comandos gráficos.

Si desea utilizarlos para confeccionar una imagen, deberá calcular previamente en un papel milimetrado todas las posiciones necesarias para la entrada del gráfico. Esto no puede evitarse, pero la confección de una imagen titular ya no presentará problemas utilizando el programa 11.

Usted se mueve por la pantalla con un retículo, y se dirige hacia el objetivo deseado sin preocuparse de las coordenadas. A continuación, este programa permite ejecutar los comandos del BASIC MSX directamente pulsando las teclas de funciones:

F1: Trazar círculo	F2: Trazar línea
F3: Cambiar color carácter	F4: Texto en gráficos
F5: Dibujar si/no	F6: Grabar la imagen
F7: Cargar una imagen	F8: Cambiar color bordes
F9: Colorear superficies	F10: Copiar partes

Sin embargo, deberá acostumbrarse a que la grabación y carga de pantallas no pueda efectuarse en fracciones de segundos, sino en minutos.

```
10 REM Programa 11
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKEft &
                           Rainer Luers
40 REM Editor, que permite crear grafi-
       cos en la pantalla, asi como gra-
       barlos, utilizando los comandos
       disponibles. Facil control a
       traves de las teclas de funcion.
50 REM Permite la grabacion en cassette
       de las pantallas, asi como copiar
       solo partes de ella. Vea el
       cuadro de comandos a partir de
       la linea 70.
60 REM Para evitar que se vuelva a la
       pantalla con una combinacion im-
       prevista de colores, una rutina
       de seguridad contra interrup-
       ciones con ON STOP GOSUB
70 ON STOP GOSUB 1990
80 STOP ON
90 REM F1 Trazar circulo, cuando el se-
          gundo cursor esta posicionado
       F2 Trazar linea, cuando el segun-
          cursor esta posicionado
       F3 Cambiar color caracteres
          Entrar numero de dos cifras
100 REM F4 Escribir texto en la posicion
           del cursor; <ENTER> significa
           fin
110 REM F5 Dibujar o no dibujar
           no dibujar = movimiento
           del cursor
        F6 Grabar con nombre y titulo
           (tarda unos 10 minutos)
        F7 Cargar pantallas
```

```
120 REM     Entrar el nombre
        F8 Cambiar color de bordes
           indicar siempre dos cifras
           (p.e. '08' pero sin pulsar
            <ENTER>
130 REM F9 Colorear superficies con
           el color actual
           Cuidado! Se colorea hasta
           llegar al mismo color del
           borde
140 REM F10 Copiar partes de la pan-
            talla. Cursor actual es la
            esquina inferior izquierda
            2.cursor=superior derecha
            3.Cursor=nueva imagen es-
              quina inferior izquierda
150 ON KEY GOSUB 550,680,810,850,930,960
,1190,1410,1440,1470
160 REM Preparativos
170 COLOR 15,4,5
180 SCREEN 2,2
190 FOR N=1 TO 10
200 KEY(N) ON
210 NEXT N
220 REM Lectura del reticulo,
        que sirve de cursor
        grafico = sprite
230 FOR N=1 TO 8
240 READ A$
250 A$="&b"+A$
260 B$=B$+CHR$(VAL(A$))
270 NEXT  N
280 SPRITE$(1)=B$
290 REM Lectura del centro del reticulo,
        para permitir que se pueda ver
        el cursor sobre cualqier fondo
```

```
300 FOR N=1 TO 8
310 READ A$
320 A$="&b"+A$
330 C$=C$+CHR$(VAL(A$))
340 NEXT  N
350 SPRITE$(2)=C$
360 REM Posicionar el cursor grafico
        en el centro de la pantalla
        color negro
370 A=128
380 B=96
390 F=1
400 COLOR F
410 PUT SPRITE 1,(A,B),1
420 PUT SPRITE 2,(A,B),15
440 IF ME=1 THEN PSET(A+4,B+4),F
450 REM Consulta la direccion a traves
        del accionamiento de las teclas
        del cursor
460 A$=INKEY$
470 IF A$="" THEN GOTO 460
480 IF ASC(A$)=28 THEN IF A<>250 THEN A=
A+1
490 IF ASC(A$)=29 THEN IF A<>-4 THEN A=A
-1
500 IF ASC(A$)=30 THEN IF B<>-4 THEN B=B
-1
510 IF ASC(A$)=31 THEN IF B<>186 THEN B=
B+1
520 PUT SPRITE 1,(A,B),1
530 PUT SPRITE 2,(A,B),15
540 GOTO 410
550 REM Trazar circulo
        ==============
560 C=A
570 D=B
```

```
580 REM Consulta la direccion a traves
        del accionamiento de las teclas
        del cursor
590 A$=INKEY$
600 IF A$="" THEN GOTO 590
610 IF ASC(A$)=28 THEN IF C<>250 THEN C=
C+1
620 IF ASC(A$)=29 THEN IF C<>-4 THEN C=C
-1
630 IF ASC(A$)=30 THEN IF D<>-4 THEN D=D
-1
640 IF ASC(A$)=31 THEN IF D<>186 THEN D=
D+1
650 IF (A$)=CHR$(13) THEN A1=ABS(A-C):A2
=ABS(B-D):IF A1>=A2 THEN CIRCLE (A+4,B+4
),A1+13,F,,,4/3:RETURN ELSE CIRCLE (A+4,
B+4),A2,F,,,4/3:RETURN
660 PUT SPRITE 1,(C,D),1
670 GOTO 590
680 REM Trazar linea
        ============
690 C=A
700 D=B
710 REM Consulta la direccion a traves
        del accionamiento de las teclas
        del cursor
720 A$=INKEY$
730 IF A$="" THEN GOTO 720
740 IF ASC(A$)=28 THEN IF C<>250 THEN C=
C+1
750 IF ASC(A$)=29 THEN IF C<>-4 THEN C=C
-1
760 IF ASC(A$)=30 THEN IF D<>-4 THEN D=D
-1
770 IF ASC(A$)=31 THEN IF D<>186 THEN D=
D+1
```

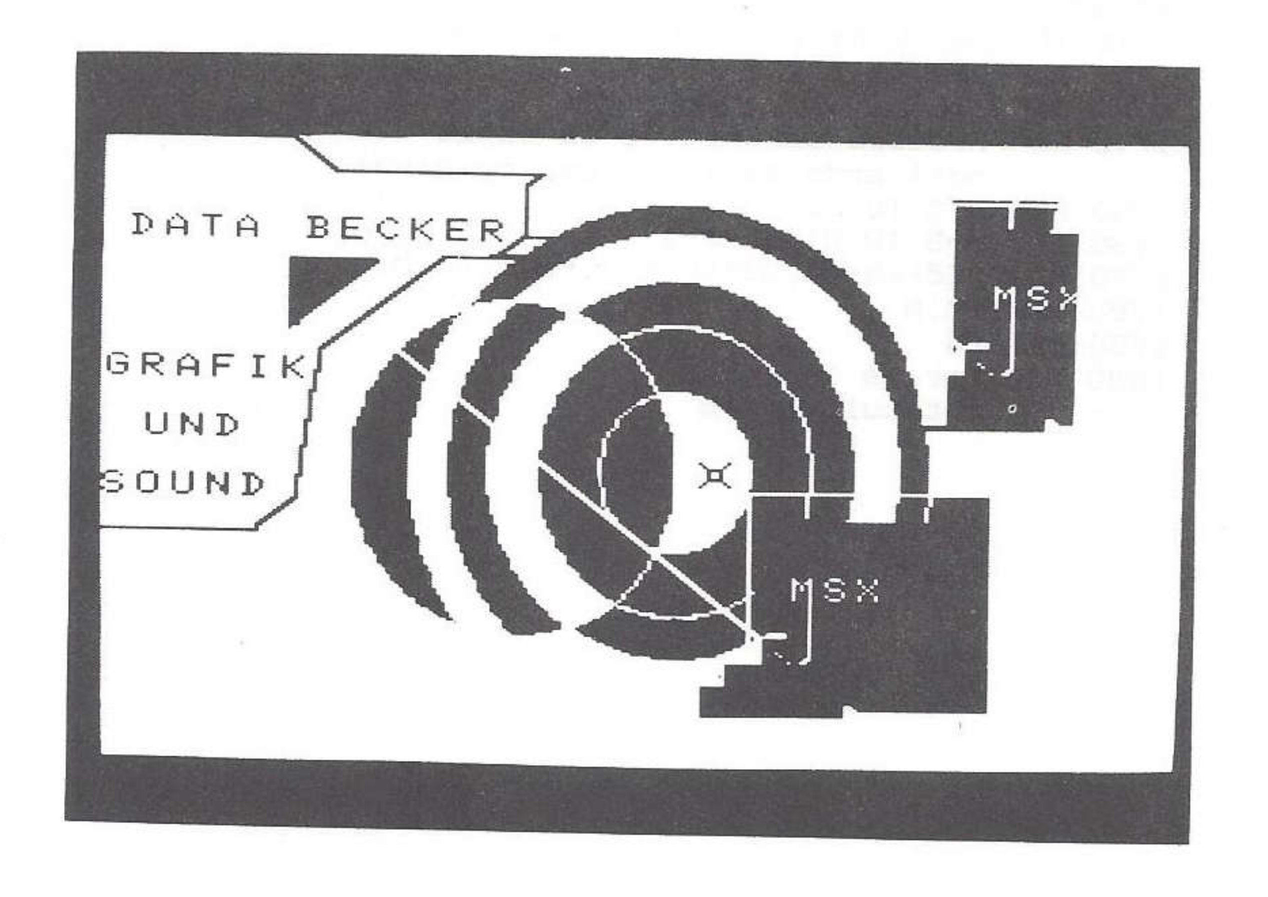
```
780 IF (A$)=CHR$(13) THEN LINE (A+4,B+4)
-(C+4,D+4),F:RETURN
790 PUT SPRITE 1,(C,D),1
800 GOTO 720
810 REM Cambiar color
        =============
820 F$=INPUT$(2)
830 F=VAL(F$)
840 IF F<16 THEN COLOR F:RETURN ELSE RET
URN
850 REM Entrar texto
        ============
860 OPEN "grp:"FOR OUTPUT AS #1
870 DRAW "bm =a;,=b;"
880 A$=INKEY$
890 IF A$="" THEN GOTO 880
900 IF A$=CHR$(13) THEN CLOSE:RETURN
910 PRINT #1,A$;
920 GOTO 880
930 REM Dibujar - no dibujar
        ====================
940 IF ME=1 THEN ME=0 ELSE ME=1
950 RETURN
960 REM Grabacion
        =========
970 REM Para evitar que se produzcan
        mezclas imprevistas de los
        colores
980 LINE (100,15)-(160,0),1,BF
990 OPEN "grp:" FOR OUTPUT AS #1
1000 NN$=""
1010 COLOR 15
1020 DRAW "bm 108,4"
1030 REM Entrar nombre del programa=
        titulo de la imagen de maximo
        seis letras/caracteres
1040 FOR N=1 TO 6
```

```
1050 NN$=NN$+N$
1060 N$=INPUT$(1)
1070 IF N$=CHR$(13) THEN GOTO 1080 ELSE
PRINT #1,N$;:NEXT N
1080 A$=INKEY$
1090 REM Funcion de seguridad
1100 IF A$="" THEN GOTO 1080 ELSE IF A$<
>CHR$(13) THEN RETURN
1110 CLOSE #1
1120 OPEN NN$ FOR OUTPUT AS #1
1130 REM Grabar los 16K
         RAM de video
1140 FOR N=0 TO 16383
1150 PRINT #1,VPEEK(N)
1160 NEXT N
1170 CLOSE
1180 RETURN
1190 REM Cargar
         ======
1200 REM Para evitar que se produzcan
         mezclas imprevistas de los
         colores
1210 LINE (100,15)-(160,0),1,BF
1220 OPEN "grp:" FOR OUTPUT AS #1
1230 NN$=""
1240 COLOR 15
1250 DRAW "bm 108,4"
1260 REM Entrar nombre del programa=
         titulo de la imagen de maximo
         seis letras/caracteres
1270 FOR N=1 TO 6
1280 NN$=NN$+N$
1290 N$=INPUT$(1)
1300 IF N$=CHR$(13) THEN GOTO 1080 ELSE
PRINT #1,N$;:NEXT N
1310 REM Funcion de seguridad
1320 A$=INKEY$
```

```
1330 IF A$="" THEN GOTO 1320 ELSE IF A$<
>CHR$(13) THEN RETURN
1340 CLOSE #1
1350 REM Cargar los 16K
       RAM de video
1360 OPEN NN$ FOR INPUT AS #1
1370 FOR N=0 TO 16383
1380 INPUT #1,D
1390 VPOKE N,D
1400 NEXT N
1410 REM Cambiar bordes
         ==============
1420 A$=INPUT$(2)
1430 IF VAL(A$)<16 THEN COLOR ,,VAL(A$):
RETURN ELSE RETURN
1440 REM Colorear
         ========
1450 PAINT (A,B),F
1460 RETURN
1470 REM Doblar
         ======
1480 C1=A
1490 D1=B
1500 REM Fijar la seccion original
         de la pantalla esquina
         superior derecha
1510 REM Consulta la direccion a traves
         del accionamiento de las teclas
         del cursor
1520 A$=INKEY$
1530 IF A$="" THEN GOTO 1520
1540 IF ASC(A$)=28 THEN IF C1<>250 THEN
C1=C1+1
1550 IF ASC(A$)=29 THEN IF C1<>-4 THEN C
1=C1-1
1560 IF ASC(A$)=30 THEN IF D1<>-4 THEN D
1=D1-1
```

```
1570 IF ASC(A$)=31 THEN IF D1<>186 THEN
D1=D1+1
1580 IF (A$)=CHR$(13) THEN GOTO 1620
1590 PUT SPRITE 1,(C1,D1),1
1600 GOTO 1520
1610 REM El cursor debe posicionarse
        arriba a la derecha en rela-
         cion al origen!
1620 IF A>C1 OR B<D1 THEN BEEP:RETURN EL
SE C2=A:D2=B
1630 REM Fijar la copia
        seccion de la pantalla
        esquina inferior izquierda
1640 REM Consulta la direccion a traves
        del accionaiento de las teclas
        del cursor
1650 A$=INKEY$
1660 IF A$="" THEN GOTO 1650
1670 IF ASC(A$)=28 THEN IF C2<>250 THEN
C2=C2+1
1680 IF ASC(A$)=29 THEN IF C2<>-4 THEN C
2=C2-1
1690 IF ASC(A$)=30 THEN IF D2<>-4 THEN D
2=D2-1
1700 IF ASC(A$)=31 THEN IF D2<>186 THEN
D2=D2+1
1710 IF (A$)=CHR$(13) THEN GOTO 1750
1720 PUT SPRITE 1,(C2,D2),1
1730 GOTO 1650
1740 REM Proceso definitivo de copia
        partiendo de la imagen original
1750 FOR N=A TO C1
1760 FOR M=B TO D1 STEP-1
1770 PSET(C2+(N-A),D2+(M-B)),POINT(N,M)
1780 NEXT M,N
1790 RETURN
1800 REM Sprite 1
        Reticulo negro
```

```
1810 DATA 10000001
1820 DATA 01000010
1830 DATA 00111100
1840 DATA 00100100
1850 DATA 00100100
1860 DATA 00111100
1870 DATA 01000010
1880 DATA 10000001
1890 REM Sprite 2
        Punto central blanco del
        reticulo
1900 DATA 00000000
1910 DATA 00000000
1920 DATA 00000000
1930 DATA 00011000
1940 DATA 00011000
1950 DATA 00000000
1960 DATA 00000000
1970 DATA 00000000
1980 REM Al interrumpir el programa se
        vuelve a crear un estado de
        colores legible
1990 COLOR 15,1
```



Descripción de programa 12

Hacia el final de este libro, además del juego de caracteres del ordenador MSX, encontrará otro, el llamado juego de caracteres en letra de ordenador. Este último puede encontrarse impreso en mi libro 'Programas y utilidades' de DATA BECKER.

En el presente libro puede adquirir muchos conocimientos sobre la memorización y modificación de los juegos de caracteres, además de contar con un lujoso generador de caracteres en el programa 14, por lo cual me pareció conveniente ampliar un poco el programa de salida por impresora e incluir su listado en este libro. De esta forma usted mismo puede modificar, por ejemplo, el juego de caracteres (a través del programa 14) y cargar a continuación el programa 12 en su ordenador (¡naturalmente, su juego de caracteres no será borrado al cargar el nuevo programa, porque la RAM de vídeo y la memoria de programas están separadas!).

Sin embargo, debe tener en cuenta que dicho programa no ha sido confeccionado para una impresora del MSX, sino para una 'no-MSX', la EPSON MX-80. Por lo tanto, el único cambio a efectuar en el programa (si posee una impresora específica para el MSX) consiste en sustituir el carácter de bloque CHR$(223) del EPSON por el carácter CHR$(219) del MSX.

La longitud del papel está graduada a la estándar de 72 líneas por página. Además, al indicar los números de los caracteres a imprimir, sólo debe entrar una cantidad divisible entre 3 (por ejemplo, de carácter 129 al 218 = 90 caracteres = 30 líneas de 3 caracteres cada una = 7.5 páginas de 12 caracteres).

Por cierto: para utilizar este programa, así como el juego de caracteres modificado, debe hallarse previamente en SCREEN 1. De lo contrario, cuando dé la instrucción SCREEN, se borrará la RAM de vídeo y con ella todo su juego de caracteres.

```
10 REM Programa 12
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                     Rainer Luers
40 REM Programa, que permite imprimir el
       juego de caracteres ampliado. El
       programa fue creado para una im-
       presora EPSON (no-MSX) y tendra
       que adaptarse en caso necesario
       (bloque con EPSON: CHR$(223),
50 REM bloque con MSX: CHR$(219)). Des-
       pues de cada 12 caracteres impre-
       sos, se produce un avance de pa-
       gina. Tambien debera adaptar la
       impresion si no trabaja con papel
       DIN A4=72 lineas.
60 REM
70 CLS
80 PRINT "Usted debe hallarse"
90 PRINT "en SCREEN(1)!"
100 PRINT
110 PRINT "Cual debe ser el primer"
120 PRINT "caracter a imprimir"
130 INPUT "(1 bis 255) ";A
140 PRINT
150 PRINT "Cual debe ser el ultimo"
160 PRINT "caracter a imprimir"
170 PRINT "(";RIGHT$(STR$(A),(LEN(STR$(A
))-1));
180 INPUT " a 255) ";B
190 REM Controlar si los caracteres es-
        cogidos caben en la impresion
        triple
200 IF A>B OR B>255 OR A<0 OR (B+1-A)/3<
>INT((B+1-A)/3) THEN RUN
210 PRINT
220 PRINT "Estos son los"
```

```
230 PRINT "caracteres a"
240 PRINT "imprimir:"
250 PRINT
260 FOR N=A TO B
270 PRINT CHR$(N);
280 NEXT N
290 PRINT
300 INPUT "Correcto  -> <ENTER> ";A$
310 IF A$<>"" THEN RUN
320 REM Posiciones de memoria en SCREEN
        1, donde se ha almacenado el as-
        pecto de los caracteres CHR$
330 FOR N=A*8 TO B*8 STEP24
340 LPRINT
350 REM Impresion del caracter izquierdo
360 LPRINT "Caracteres";INT(N/8);
370 REM Impresion del caracter central
380 LPRINT TAB(29)"Caracteres";INT((N+8)
/8);
390 REM Impresion del caracter derecho
400 LPRINT TAB(58)"Caracteres";INT((N+16
)/8)
410 LPRINT
420 LPRINT TAB(8)"76543210";
430 LPRINT TAB(37);"76543210";
440 LPRINT TAB(66);"76543210"
450 LPRINT
460 FOR M=0 TO 7
470 REM Convertir representacion decimal
        a binaria de 8 bits completos
480 A$=BIN$(VPEEK(N+M))
490 B$=BIN$(VPEEK(N+M+8))
500 C$=BIN$(VPEEK(N+M+16))
510 A$=STRING$(8-LEN(A$),"0")+A$
520 B$=STRING$(8-LEN(B$),"0")+B$
530 C$=STRING$(8-LEN(C$),"0")+C$
540 LPRINT TAB(5)M;TAB(7);
```

```
550 FOR L=1TO8
560 REM Convertir representacion binaria
        a representacion de espacios-
        bloques de la impresora
570 IF MID$(A$,L,1)="1" THEN LPRINT CHR$
(223);ELSE LPRINT" ";
580 NEXT L
590 LPRINT TAB(34)M;TAB(36);
600 FOR L=1TO8
610 IF MID$(B$,L,1)="1" THEN LPRINT CHR$
(223);ELSE LPRINT" ";
620 NEXT L
630 LPRINT TAB(63)M;TAB(65);
640 FOR L=1TO8
650 IF MID$(C$,L,1)="1" THEN LPRINT CHR$
(223);ELSE LPRINT" ";
660 NEXT L
670 LPRINT
680 NEXT M
690 LPRINT
700 REM Controlar si ya es preciso
        efectuar un avance de pagina
710 Z=Z+1
720 IF Z=4 THEN Z=0:FOR U=1 TO 16:LPRINT
:NEXT
730 NEXT N
740 RUN
```

```
Usted debe hallarse
en SCREEN(1)!

Cual debe ser el primer
caracter a imprimir
(1 a 255) ?  33

Cual debe ser el ultimo
caracter a imprimir
(33 a 255) ?  128

Estos son los
caracteres a
imprimir:

!"#$%&'()*+,-./0123456789:;<=
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]^_`abcdefghijklmnopqrstuvw
xyz{|}~Ç

Correcto -> <ENTER> ?
```

Descripción de programa 13

El ordenador MSX dispone de dos pantallas de texto (SCREEN 0 de 40 caracteres por línea, y SCREEN 1 de 32 caracteres por línea) y de dos pantallas gráficas (SCREEN 2 de 256*192 puntos, y SCREEN 3 de 64*48 puntos).

Es posible emitir texto en las pantallas gráficas, pero los gráficos sólo pueden representarse en pantallas de texto cuando se trate del control de sprites en SCREEN 1.

El siguiente programa pone fin a la separación de textos y gráficos. ¡Bueno, tampoco es tan fácil!

En el programa 13 hemos empleado algunos trucos de la siguiente forma: en SCREEN 0 y SCREEN 1 (los modos de texto) puede visualizar simultáneamente hasta 256 caracteres diferentes. Si usted modifica el aspecto de uno de estos caracteres, todos aquellos caracteres con idéntico código también serán modificados en la pantalla. Puede cambiar el aspecto de cualquiera de los 256 caracteres a conveniencia. Hasta aquí las condiciones previas (consulte los capítulos CHR$ y ASC de este libro).

A través del programa 13 se efectúa ahora una continua modificación de una cantidad máxima de 4 caracteres, de modo que parezca como si un pequeño gráfico se fuera desplazando ante sus ojos por la pantalla de texto. Incluso hemos culminado el programa escribiendo en la pantalla las iniciales de DATA BECKER a través de 4 caracteres ...

```
10 REM Programa 13
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                        Rainer Luers
40 REM Este programa le muestra como
       puede crear graficos en SCREEN 1
       (tambien posible en SCREEN 0, si
       consideramos la direccion inicial
       de los otros caracteres en la
50 REM VRAM), aunque, en principio, este
       modo no sea capaz de generar
       caracteres graficos si no son de
       definicion previa o propia. Para
       obtener mas informacion sobre
60 REM este fenomeno, lea el texto ex-
       plicativo del programa.
70 REM Preparativos
80 SCREEN 1
90 KEY OFF
100 REM Borrar caracter 250
110 REM 1) un punto cambia continuamente
           su posicion
120 GOSUB 980
130 REM se posiciona CHR$(250) arriba a
        la izquierda
140 LOCATE 0,0
150 PRINT CHR$(250)
160 LOCATE 6,10
170 PRINT "SCREEN 0 con HGR 1"
180 FOR L=1 TO 10
190 REM Movimiento del punto
200 GOSUB 870
210 REM Borrar caracter CHR$ 250
220 GOSUB 980
230 NEXT L
240 REM 2) muchos puntos cambian su po-
           sicion de forma regularmente
           continua
```

```
250 GOSUB 1300
260 REM En toda la pantalla se posiciona
        el caracter CHR$ 250
270 LOCATE 6,10
280 PRINT "SCREEN 0 con HGR 1a"
290 FOR L=1 TO 10
300 REM Movimiento del punto
310 GOSUB 870
320 REM Borrar caracter CHR$ 250
330 GOSUB 980
340 NEXT L
350 REM 3) Dibujo regularmente continuo
           con un caracter CHR$ 250
360 CLS
370 LOCATE 0,0
380 PRINT CHR$(250)
390 LOCATE 6,10
400 PRINT "SCREEN 1 con HGR 2"
410 FOR L=1 TO 10
420 REM Desarrollo del dibujo
430 GOSUB 1030
440 REM Borrar caracter
450 GOSUB 980
460 NEXT L
470 REM 4) Dibujo regularmente continuo
           en multiples posiciones
480 GOSUB 1300
490 REM En toda la pantalla se posiciona
        el caracter CHR$ 250
500 LOCATE 6,10
510 PRINT "SCREEN 0 con HGR 2a"
520 FOR L=1 TO 10
530 REM Desarrollo del dibujo
540 GOSUB 1030
550 REM Borrar caracter
560 GOSUB 980
```

```
570 NEXT L
580 REM 5) Se recurre a un bloque de ca-
           racteres (compuesto por CHR$
           (250) a CHR$(253)) para re-
           presentar grafica continua
590 CLS
600 LOCATE 0,0
610 REM Borrar caracter
620 GOSUB 1230
630 PRINT CHR$(250);CHR$(252)
640 PRINT CHR$(251);CHR$(253)
650 LOCATE 6,10
660 PRINT "SCREEN 1 con HGR 3"
670 FOR L=1 TO 10
680 REM Leer y representar caracter
690 GOSUB 1120
700 REM Borrar caracter
710 GOSUB 1230
720 NEXT L
730 REM 6) Se utilizan muchos bloques de
           caracteres (formados cada uno
           por CHR$(250) a CHR$(253)),
           para representar graficas
           continuas
740 CLS
750 REM LLenar la pantalla de los carac-
        teres CHR$ 250 a 253
760 GOSUB 1370
770 LOCATE 6,10
780 PRINT "SCREEN 1 con HGR 3a"
790 FOR L=1 TO 10
800 REM Leer y representar caracter
810 GOSUB 1120
820 REM Borrar caracter
830 GOSUB 1230
840 NEXT L
850 END
```

```
860 REM Subprograma para leer informa-
        cion sobre el aspecto del carac-
        ter de CHR$(250) (Representacion
        puntual, vea 1 y 2)
870 RESTORE 1280
880 FOR N=1 TO 20
890 READ A,B
900 VPOKE(250*8+A),B
910 REM Volver a borrar punto
920 FOR M=1 TO 10
930 NEXT M
940 VPOKE(250*8+A),0
950 NEXT N
960 RETURN
970 REM Subprograma borrado caracteres
980 FOR N=250*8+0 TO 250*8+7
990 VPOKE N,0
1000 NEXT N
1010 RETURN
1020 REM Subprograma lectura de informa-
         cion sobre el aspecto del ca-
         racter CHR$(250) (dibujos en
         CHR$(250), vea 3 y 4)
1030 RESTORE 1280
1040 FOR N=1 TO 20
1050 READ A,B
1060 B=VPEEK(250*8+A) OR B
1070 VPOKE(250*8+A),B
1080 FOR M=1 TO 10
1090 NEXT M
1100 NEXT N
1110 RETURN
1120 REM Subprograma lectura de informa-
         cion sobre 4 caracteres al
         mismo tiempo (graficos en
         CHR$(250) a CHR$(253),
         vea 5 y 6)
```

```
1130 RESTORE 1490
1140 FOR N=1 TO 73
1150 READ A,B,C
1160 C=VPEEK(250*8+A*8+B) OR C
1170 VPOKE (250*8+A*8+B),C
1180 FOR M=1 TO 10
1190 NEXT M
1200 NEXT N
1210 RETURN
1220 REM Subprograma borrado de caracte-
        res (CHR$(250) a CHR$(253))
1230 FOR N=250*8 TO 250*8+32
1240 VPOKE N,0
1250 NEXT N
1260 RETURN
1270 REM DATAs sobre aspecto de
        CHR$(250) en partes 1 a 4
1280 DATA 0,128,1,64,2,32,3,16,4,16,5,32
,6,32,7,16,6,8,6,4,6,2,5,1,4,1,3,2,2,4,1
,4,0,8,0,16,0,32,0,64
1290 REM Subprograma llenado de la pan-
        talla con CHR$(250)
1300 FORN=0 TO 28
1310 FOR M=0 TO 23
1320 LOCATE N,M
1330 PRINT CHR$(250);
1340 NEXT M,N
1350 RETURN
1360 REM Subprograma llenado de la pan-
        talla con bloques de caracteres
        (CHR$(250) a CHR$(253))
1370 FOR L=1 TO 12
1380 FOR N=0 TO 13
1390 PRINT CHR$(250);CHR$(252);
1400 NEXT N
1410 PRINT
1420 FOR N=0 TO 13
```

```
1430 PRINT CHR$(251);CHR$(253);
1440 NEXT N
1450 PRINT
1460 NEXT L
1470 RETURN
1480 REM DATAs sobre aspecto de
         CHR$(250) a CHR$(253)
1490 DATA 0,2,16,0,3,16,0,4,16,0,5,16,0,
6,16,0,7,16,1,0,16,1,1,16,1,2,16,1,3,16,
1,4,32,1,4,64,1,5,128,1,6,64,1,5,32,1,4,
16,1,5,16,1,6,8,1,5,4,1,4,2,1,3,2,1,2,1,
1,1,1,1,0,1,0,7,1,0,6,1,0,5,1,0,4,1,0,3,
1
1500 DATA 0,2,2,0,1,4,0,1,8,0,1,16,0,1,3
2,0,1,64,0,2,128,0,3,128
1510 DATA 2,2,64,2,3,64,2,4,64,2,5,64,2,
6,64,2,7,64,3,0,64,3,1,64,3,2,64,3,3,64,
3,4,64,3,5,64,3,6,64,2,2,128,2,1,64,2,1,
32,2,1,16,2,1,8,2,1,4,2,2,2,2,3,1,2,4,1,
2,5,2,2,6,4,2,7,8,3,0,4,3,1,2,3,2,1
1520 DATA 3,3,1,3,4,2,3,5,4,3,5,8,3,6,16
,3,6,32,3,6,64,3,6,128
```

Descripción de programa 14

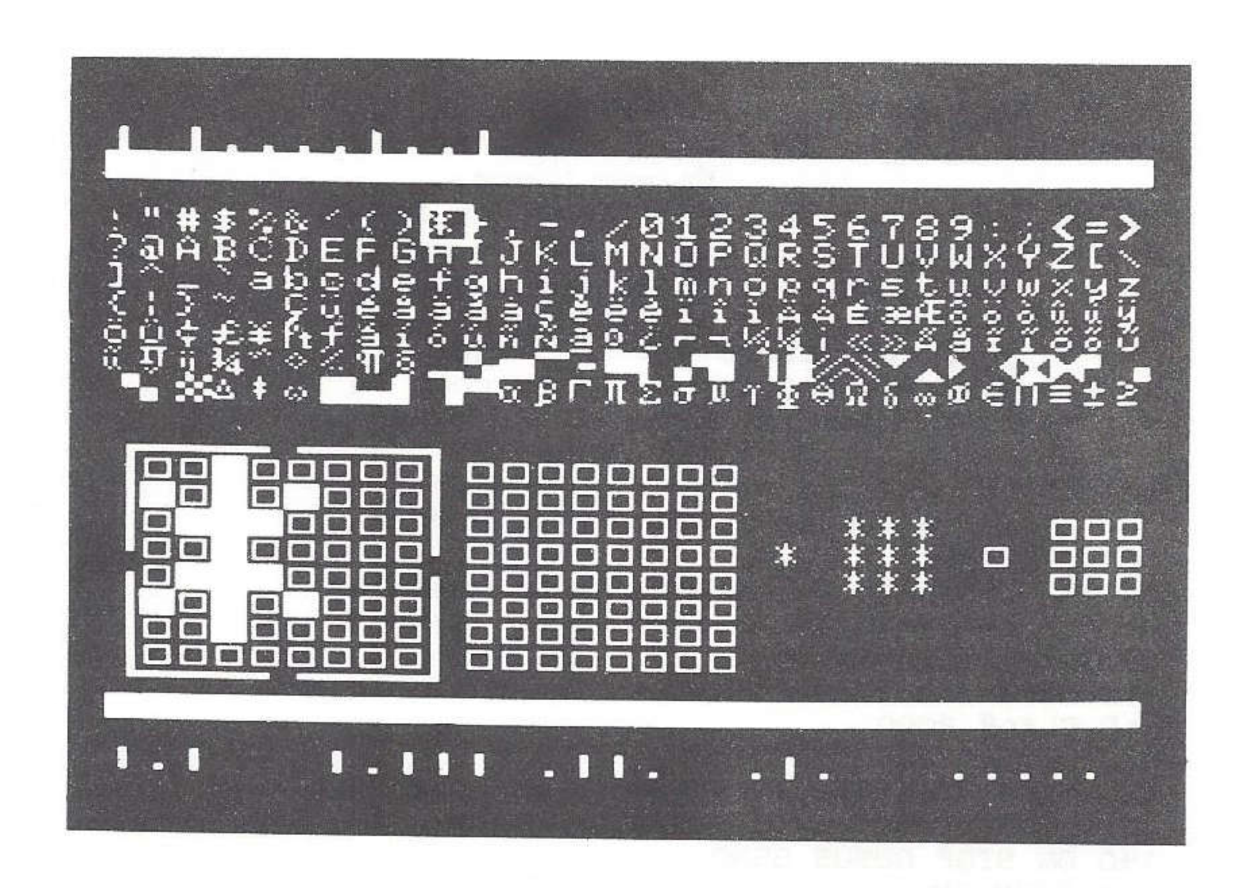
Este programa permite modificar el juego de caracteres del ordenador MSX pasando de los comandos VPEEK y VPOKE.
Sin embargo, al planificar el programa tropezamos con un problema: si pueden modificarse todos los caracteres (p.e. si puede memorizar una 'B' en lugar de la 'A', o también un '2' en lugar del '1'), también se modificarían los correspondientes textos auxiliares o números del mismo programa. Por lo tanto, se ha creado un juego de caracteres, o mejor dicho un lenguaje por imágenes, compuesto por sólo 3 caracteres: espacio, barra vertical y punto. Estos tres caracteres no podrán ser modificados mediante el programa 14. ¿Cómo funciona ahora este lenguaje por imágenes para emitir, por ejemplo, un número?
El punto sirve para indicar las unidades, decenas y centenas; la barra separa las unidades de las decenas y éstas, por su parte, de las centenas. El espacio indica una cantidad vacía. El número 214 se entraría de la forma siguiente:

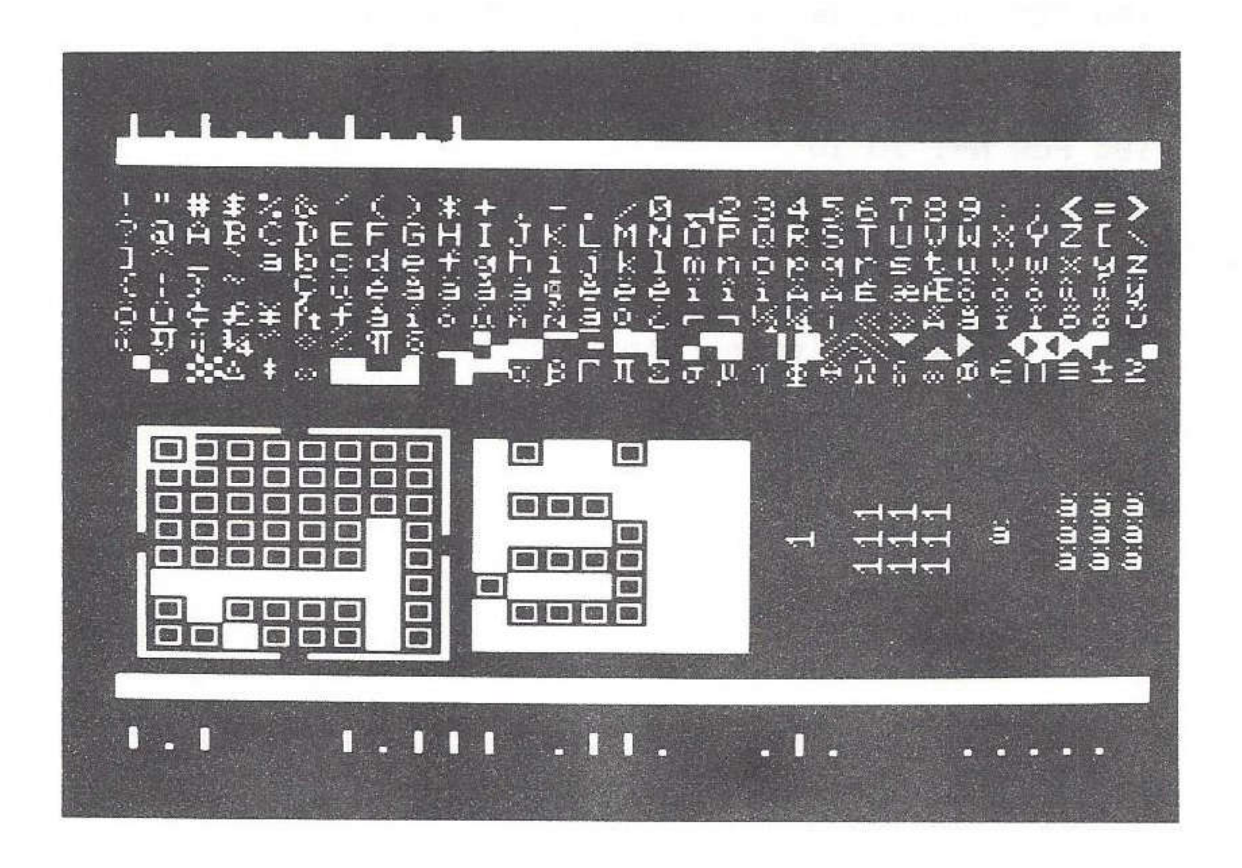
2 centenas = 2 puntos. Una barra de separación.
1 decena = 1 punto. Una barra de separación.
4 unidades = 4 puntos. Una barra de separación final.

Las teclas de funciones tienen los significados siguientes:

F1/F6: barra, punto, barra: cargar/grabar juego de caract.
F2/F7: 3 barras, 1 punto, 1 barra: bloque de caracteres 1/2
F3: punto, 2 barras, punto: reflexión
F4: punto, barra, punto: giro hacia la derecha
F5: 5 puntos: validar/invalidar 'Help' (ayuda)
F8: barra, espacio, barra: borrar
F9: punto, barra, espacio, barra: representación inversa
F10: espacio, barra, espacio, 2 barras: cambio

Con la tecla tabuladora va hacia el editor (con barra espaciadora borrar y colocar puntos), con ENTER da su aprobación al cambio que ha efectuado del carácter, y ESC le devuelve al juego de caracteres.





```
10 REM Programa 14
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
                     Rainer Luers
40 REM Generador de caracteres, que le
       facilita la creacion de sus pro-
       pios juegos de caracteres, asi
       como modificar parcialmente el
       juego existente del MSX. Diez
50 REM diversas funciones, tales como la
       representacion invertida y sime-
       trica, le faclitan la tarea.
60 REM La pantalla de trabajo comienza
       en la posicion 6144 de la VRAM
70 VDP(2)=6
80 REM El juego de caracteres de trabajo
       comienza en posicion 0 de la VRAM
90 VDP(4)=0
100 DEFINT A-Z
110 CLEAR 2000
120 DIM H$(13)
130 REM Imposible interrumpir el progra-
        ma con pantalla desordenada
140 ON STOP GOSUB 5530
150 STOP ON
160 REM Inicio de los subprogramas para
        diversas funciones
170 ON KEY GOSUB 3600,3710,3800,4180,456
0,4590,4700,4790,5070,5350
180 FOR N=1 TO 10
190 KEY(N) ON
200 NEXT N
210 REM Definir los caracteres CHR$ 251
        a 254 para la indicacion de las
        teclas de funcion y para crear
        numeros, entre otros
220 DATA 00000000
```

```
230 DATA 01111110
240 DATA 01000010
250 DATA 01000010
260 DATA 01000010
270 DATA 01000010
280 DATA 01111110
290 DATA 00000000
300 '
310 DATA 11111111
320 DATA 11111111
330 DATA 11111111
340 DATA 11111111
350 DATA 11111111
360 DATA 11111111
370 DATA 11111111
380 DATA 11111111
390 '
400 DATA 00000000
410 DATA 00000000
420 DATA 00000000
430 DATA 00000000
440 DATA 00000000
450 DATA 00000000
460 DATA 00011000
470 DATA 00011000
480 '
490 DATA 00000000
500 DATA 00011000
510 DATA 00011000
520 DATA 00011000
530 DATA 00011000
540 DATA 00011000
550 DATA 00011000
560 DATA 00011000
570 '
580 REM Aspecto de cuatro sprites 16*16
        para ribetear bloques
```

```
590 DATA 1111111111111111
600 DATA 1000000000000000
610 DATA 1000000000000000
620 DATA 1000000000000000
630 DATA 1000000000000000
640 DATA 1000000000000000
650 DATA 1000000000000000
660 DATA 1000000000000000
670 DATA 1000000000000000
680 DATA 1000000000000000
690 DATA 1000000000000000
700 DATA 1000000000000000
710 DATA 1000000000000000
720 DATA 1000000000000000
730 DATA 1000000000000000
740 DATA 1000000000000000
750 '
760 DATA 1111111111111111
770 DATA 0000000000000001
780 DATA 0000000000000001
790 DATA 0000000000000001
800 DATA 0000000000000001
810 DATA 0000000000000001
820 DATA 0000000000000001
830 DATA 0000000000000001
840 DATA 0000000000000001
850 DATA 0000000000000001
860 DATA 0000000000000001
870 DATA 0000000000000001
880 DATA 0000000000000001
890 DATA 0000000000000001
900 DATA 0000000000000001
910 DATA 0000000000000001
920 '
930 DATA 1000000000000000
940 DATA 1000000000000000
950 DATA 1000000000000000
```

```
960 DATA 1000000000000000
970 DATA 1000000000000000
980 DATA 1000000000000000
990 DATA 1000000000000000
1000 DATA 1000000000000000
1010 DATA 1000000000000000
1020 DATA 1000000000000000
1030 DATA 1000000000000000
1040 DATA 1000000000000000
1050 DATA 1000000000000000
1060 DATA 1000000000000000
1070 DATA 1000000000000000
1080 DATA 1111111111111111
1090 '
1100 DATA 0000000000000001
1110 DATA 0000000000000001
1120 DATA 0000000000000001
1130 DATA 0000000000000001
1140 DATA 0000000000000001
1150 DATA 0000000000000001
1160 DATA 0000000000000001
1170 DATA 0000000000000001
1180 DATA 0000000000000001
1190 DATA 0000000000000001
1200 DATA 0000000000000001
1210 DATA 0000000000000001
1220 DATA 0000000000000001
1230 DATA 0000000000000001
1240 DATA 0000000000000001
1250 DATA 1111111111111111
1260 '
1270 REM Sprite para escoger y editar
        CHR$
1280 DATA 1111110000000000
1290 DATA 1000010000000000
1300 DATA 1000010000000000
1310 DATA 1000010000000000
```

```
1320 DATA 1000010000000000
1330 DATA 1111110000000000
1340 DATA 0000000000000000
1350 DATA 0000000000000000
1360 DATA 0000000000000000
1370 DATA 0000000000000000
1380 DATA 0000000000000000
1390 DATA 0000000000000000
1400 DATA 0000000000000000
1410 DATA 0000000000000000
1420 DATA 0000000000000000
1430 DATA 0000000000000000
1440 '
1450 REM Codificacion de las diez
         teclas de funcion
1460 KEY 1,CHR$(254)+CHR$(253)+CHR$(254)
1470 KEY 2,CHR$(254)+CHR$(253)+CHR$(254)
+CHR$(254)+CHR$(254)
1480 KEY 3,CHR$(253)+CHR$(254)+CHR$(254)
+CHR$(253)
1490 KEY 4,CHR$(253)+CHR$(254)+CHR$(253)
1500 KEY 5,STRING$(5,CHR$(253))
1510 KEY 6,CHR$(254)+CHR$(253)+CHR$(253)
+CHR$(254)
1520 KEY 7,CHR$(254)+CHR$(254)+CHR$(254)
+CHR$(253)+CHR$(254)
1530 KEY 8,CHR$(254)+" "+CHR$(254)
1540 KEY 9,CHR$(253)+CHR$(254)+" "+CHR$(
254)
1550 KEY 10," "+CHR$(254)+" "+CHR$(254)+
CHR$(254)
1560 REM Activar SCREEN 1 con repre-
         sentacion ampliada de
         sprites 16*16
1570 SCREEN 1,3
1580 COLOR 15,1
1590 WIDTH 30
```

```
1600 KEY ON
1610 PRINT "Un momento, por favor ..."
1620 REM Cargar el codigo de caracteres
        en caracteres CHR$ 251 a 254
1630 FOR N=0 TO 3
1640 FOR M=0 TO 7
1650 READ A$
1660 A=VAL("&b"+A$)
1670 VPOKE (BASE(7)+((251+N)*8))+M,A
1680 NEXT M
1690 NEXT N
1700 REM Copiar el juego de caracteres
        en las posiciones 10240 y si-
        guientes de la VRAM (VDP(4)=5))
1710 FOR N%=0 TO 2048
1720 VPOKE N%+10240,VPEEK(N%)
1730 NEXT N%
1740 REM Grabar una segunda pantalla
        como pantalla auxiliar
        (VDP(2)=9)
1750 H$(0)="     contenido teclas de func
ion"
1760 H$(1)="     "+STRING$(23,219)
1770 H$(2)=STRING$(32," ")
1780 H$(3)=STRING$(32," ")
1790 H$(4)="     1 = cargar juego caracte
res"
1800 H$(5)="     2 = bloque caracteres 1"
1810 H$(6)="     3 = simetria"
1820 H$(7)="     4 = giro hacia derecha"
1830 H$(8)="     8 = ayuda activada/desac
tivada"
1840 H$(9)="     6 = grabar juego caracte
res"
1850 H$(10)="     7 = bloque caracteres 2
"
1860 H$(11)="     8 = borrar"
1870 H$(12)="     9 = representacion inve
rtida"
1880 H$(13)="    10 = invertir 1 - 2"
```

```
1890 REM Borrar la segunda pantalla
1900 FOR N=9216 TO 9216+767
1910 VPOKE N,32
1920 NEXT N
1930 REM Visualizar lineas auxiliares
1940 FOR N=0 TO 13
1950 FOR M=1 TO LEN(H$(N))
1960 VPOKE 9215+(N*32+M),ASC(MID$(H$(N),
M,1))
1970 NEXT M
1980 NEXT N
1990 REM Cargar los 5 sprites de las
       lineas DATA
2000 FOR N=1 TO 5
2010 S$=SPACE$(32)
2020 FOR M=1 TO 16
2030 READ A$
2040 MID$(S$,M,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
2050 MID$(S$,M+16,1)=CHR$(VAL("&b"+RIGHT
$(A$,8)))
2060 NEXT M
2070 SPRITE$(N)=S$
2080 NEXT N
2090 REM Inicio visible del programa
2100 CLS
2110 PRINT
2120 PRINT STRING$(30,252);
2130 PRINT
2140 REM Visualizar los caracteres CHR$
       33 a 126
2150 FOR N=33 TO 126
2160 PRINT CHR$(N);
2170 NEXT N
2180 PRINT " ";
2190 REM Visualizar los caracteres CHR$
       128 a 242
```

```
2200 FOR N=128 TO 242
2210 PRINT CHR$(N);
2220 NEXT N
2230 PRINT
2240 PRINT
2250 REM Visualizar los 2 bloques
        editores
2260 FOR N=1 TO 8
2270 PRINT " ";STRING$(8,CHR$(251));" ";
STRING$(8,CHR$(251))
2280 NEXT N
2290 REM Visualizar siempre 10 caracte-
        res, que sirven de control
        de edicion
2300 LOCATE 19,15
2310 PRINT CHR$(251);
2320 LOCATE 25,15
2330 PRINT CHR$(251);
2340 FOR N=21 TO 23
2350 FOR M=14 TO 16
2360 LOCATE N,M
2370 PRINT CHR$(251);
2380 NEXT M,N
2390 FOR N=27 TO 29
2400 FOR M=14 TO 16
2410 LOCATE N,M
2420 PRINT CHR$(251);
2430 NEXT M,N
2440 LOCATE 0,21
2450 PRINT STRING$(30,CHR$(252));
2460 REM Dibujar los 4 sprites de bloque
        y el sprite de edicion
2470 PUT SPRITE 5,(6,21),15
2480 PUT SPRITE 1,(13,92),15
2490 PUT SPRITE 2,(51,92),15
2500 PUT SPRITE 3,(13,130),15
2510 PUT SPRITE 4,(51,130),15
```

```
2520 REM Punto de salida del sprite del
         cursor
2530 Z1=6
2540 Z2=21
2550 C1=33
2560 REM Control de consulta del teclado
2570 A$=INKEY$
2580 IF A$="" THEN GOTO 2570
2590 REM ESC -> Cambiar indicacion CHR$
         al editor de bloques
2600 IF ASC(A$)=27 THEN ED=1:GOTO 3210
2610 REM Lectura teclas del cursor
2620 IF ASC(A$)=28 THEN IF (Z1<>238) THE
N  Z1=Z1+8:PUT SPRITE 5,(Z1,Z2),15:C1=C1
+1:GOTO 3120
2630 IF ASC(A$)=29 AND (Z1<>6) THEN  Z1=
Z1-8:PUT SPRITE 5,(Z1,Z2),15:C1=C1-1:GOT
O 3120
2640 IF ASC(A$)=30 AND (Z2<>21) THEN Z2=
Z2-8:PUT SPRITE 5,(Z1,Z2),15:C1=C1-30:GO
TO 3120
2650 IF ASC(A$)=31 AND (Z2<>69) THEN Z2=
Z2+8:PUT SPRITE 5,(Z1,Z2),15:C1=C1+30:GO
TO 3120
2660 REM TAB -> representacion ampliada
         del caracter escogido en el
         bloque actual
2670 IF ASC(A$)=9 THEN IF M1=2 THEN GOTO
 2900 ELSE GOTO 2690 ELSE GOTO 2570
2680 REM Bloque izquierdo (M1=1)
2690 FOR N=0 TO 7
2700 A$(N)=BIN$(VPEEK(BASE(7)+(C1*8)+N))
2710 A$(N)=STRING$(8-LEN(A$(N)),"0")+A$(
N)
2720 NEXT N
2730 FOR N=0 TO 7
2740 FOR M=0 TO 7
```

```
2750 LOCATE M+1,N+12
2760 IF MID$(A$(N),M+1,1)="1" THEN PRINT
 CHR$(252); ELSE PRINT CHR$(251);
2770 A(M+1,N+1)=VAL(MID$(A$(N),M+1,1))
2780 NEXT M
2790 NEXT N
2800 LOCATE 19,15
2810 PRINT CHR$(C1);
2820 FOR N=21 TO 23
2830 FOR M=14 TO 16
2840 LOCATE N,M
2850 PRINT CHR$(C1);
2860 NEXT M
2870 NEXT N
2880 GOTO 2570
2890 REM bloque derecho (M1=2)
2900 FOR N=0 TO 7
2910 B$(N)=BIN$(VPEEK(BASE(7)+(C1*8)+N))
2920 B$(N)=STRING$(8-LEN(B$(N)),"0")+B$(
N)
2930 NEXT N
2940 FOR N=0 TO 7
2950 FOR M=0 TO 7
2960 LOCATE M+10,N+12
2970 IF MID$(B$(N),M+1,1)="1" THEN PRINT
 CHR$(252); ELSE PRINT CHR$(251);
2980 B(M+1,N+1)=VAL(MID$(B$(N),M+1,1))
2990 NEXT M
3000 NEXT N
3010 LOCATE 25,15
3020 PRINT CHR$(C1);
3030 FOR N=27 TO 29
3040 FOR M=14 TO 16
3050 LOCATE N,M
3060 PRINT CHR$(C1);
3070 NEXT M
3080 NEXT N
```

```
3090 REM Indicacion del numero de carac-
         ter CHR$ con puntos y rayas
         (CHR$(253) y CHR$(254))
3100 GOTO 2570
3110 NEXT N
3120 LOCATE 0,0
3130 PRINT SPACE$(23);
3140 LOCATE 0,0
3150 A$=CHR$(254)
3160 IF C1>99 THEN Z=INT(C1/100):A$=A$+S
TRING$(Z,CHR$(253))+CHR$(254):C2=C1-(Z*1
00) ELSE A$=A$+" "+CHR$(254):C2=C1
3170 IF C2>9 THEN Z=INT(C2/10):A$=A$+STR
ING$(Z,CHR$(253))+CHR$(254):C3=C2-(Z*10)
 ELSE A$=A$+" "+CHR$(254):C3=C2
3180 IF C3>0 THEN Z=C3:A$=A$+STRING$(Z,C
HR$(253))+CHR$(254) ELSE A$=A$+" "+CHR$(
254)
3190 PRINT A$;
3200 GOTO 2570
3210 REM Trabajo como editor con repre-
         sentacion ampliada de CHR$
3220 IF M1=2 THEN GOTO 3420
3230 REM Trabajo en el bloque izquierdo
         (M1=1)
3240 M5=14
3250 M6=93
3260 M7=1
3270 M8=1
3280 PUT SPRITE 5,(M5,M6),15
3290 A$=INKEY$:IF A$="" THEN GOTO 3290
3300 REM ENTER -> Transferir la funcion
         editora al caracter CHR$
3310 IF ASC(A$)=13 THEN FOR N=0 TO 7:FOR
 M=1 TO 8:Q$=Q$+RIGHT$(STR$(A(M,N+1)),1)
:NEXT M:VPOKE C1*8+N,VAL("&b"+Q$):Q$="":
NEXT N
```

```
3320 REM Espacio -> Posicionar pixel
        o bien borrarlo
3330 IF A$=" " THEN LOCATE O+M7,11+M8:IF
 A(M7,M8)=1 THEN PRINT CHR$(251);:A(M7,M
8)=0 ELSE PRINT CHR$(252);:A(M7,M8)=1
3340 REM ESC -> Cambio del editor de
        bloques a indicacion CHR$
3350 IF ASC(A$)=27 THEN ED=0:PUT SPRITE
5,(Z1,Z2),15:GOTO 2570
3360 REM Lectura teclas de funcion
3370 IF ASC(A$)=28 AND M7<>8 THEN M5=M5+
8:M7=M7+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290
3380 IF ASC(A$)=29 AND M7<>1 THEN M5=M5-
8:M7=M7-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290
3390 IF ASC(A$)=31 AND M8<>8 THEN M6=M6+
8:M8=M8+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290
3400 IF ASC(A$)=30 AND M8<>1 THEN M6=M6-
8:M8=M8-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290 ELSE GOTO 3290
3410 REM Trabajo en el bloque derecho
        (M1=2)
3420 M5=86
3430 M6=93
3440 M7=1
3450 M8=1
3460 PUT SPRITE 5,(M5,M6),15
3470 A$=INKEY$
3480 IF A$="" THEN GOTO 3470
3490 REM ENTER -> Transferir la funcion
        editora al caracter CHR$
3500 IF ASC(A$)=13 THEN FOR N=0 TO 7:FOR
 M=1 TO 8:Q$=Q$+RIGHT$(STR$(B(M,N+1)),1)
:NEXT M:VPOKE C1*8+N,VAL("&b"+Q$):Q$="":
NEXT N
3510 REM Espacio -> Posicionar pixel
        o bien borrarlo
```

```
3520 IF A$=" " THEN LOCATE 9+M7,11+M8:IF
 B(M7,M8)=1 THEN PRINT CHR$(251);:B(M7,M
8)=0 ELSE PRINT CHR$(252);:B(M7,M8)=1
3530 REM ESC -> Cambio del editor de
        bloques a indicacion CHR$
3540 IF ASC(A$)=27 THEN ED=0:PUT SPRITE
5,(Z1,Z2),15:GOTO 2570
3550 REM Lectura teclas de funcion
3560 IF ASC(A$)=28 AND M7<>8 THEN M5=M5+
8:M7=M7+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
470
3570 IF ASC(A$)=29 AND M7<>1 THEN M5=M5-
8:M7=M7-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
470
3580 IF ASC(A$)=31 AND M8<>8 THEN M6=M6+
8:M8=M8+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
470
3590 IF ASC(A$)=30 AND M8<>1 THEN M6=M6-
8:M8=M8-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
470 ELSE GOTO 3470
3600 REM Tecla de funcion 1
        cargar juego de caracteres
3610 LOCATE 10,0
3620 OPEN "charac" FOR INPUT AS #1
3630 FOR N=33 TO 249
3640 FOR M=0 TO 7
3650 INPUT #1,A%
3660 VPOKE(BASE(7)+(N*8)+M),A%
3670 NEXT M
3680 NEXT N
3690 CLOSE
3700 RETURN
3710 REM Tecla de funcion 2
        imposible saltar al bloque
3720 REM izquierdo, si actualmente se
        encuentra en un bloque editor
        (ED=1)
3730 IF ED=1 THEN RETURN
```

```
3740 PUT SPRITE 1,(13,92),15
3750 PUT SPRITE 2,(51,92),15
3760 PUT SPRITE 3,(13,130),15
3770 PUT SPRITE 4,(51.130),15
3780 M1=1
3790 RETURN
3800 REM Tecla de funcion 3
         simetria de caracteres
3810 IF M1=2 THEN GOTO 4010
3820 REM Trabajo en el bloque izquierdo
         (M1=1)
3830 FOR N=1 TO 8
3840 FOR M=1 TO 8
3850 C(N,M)=A(ABS(N-9),M)
3860 NEXT M
3870 NEXT N
3880 FOR N=1 TO 8
3890 FOR M=1 TO 8
3900 A(N,M)=C(N,M)
3910 NEXT M
3920 NEXT N
3930 FOR N=0 TO 7
3940 FOR M=0 TO 7
3950 LOCATE N+1,M+12
3960 IF A(N+1,M+1)=1 THEN PRINT CHR$(252
);ELSE PRINT CHR$(251);
3970 NEXT M
3980 NEXT N
3990 RETURN
4000 REM Trabajo en el bloque derecho
         (M1=2)
4010 FOR N=1 TO 8
4020 FOR M=1 TO 8
4030 C(N,M)=B(ABS(N-9),M)
4040 NEXT M
4050 NEXT N
4060 FOR N=1 TO 8
4070 FOR M=1 TO 8
4080 B(N,M)=C(N,M)
```

```
4090 NEXT M
4100 NEXT N
4110 FOR N=0 TO 7
4120 FOR M=0 TO 7
4130 LOCATE N+10,M+12
4140 IF B(N+1,M+1)=1 THEN PRINT CHR$(252
);ELSE PRINT CHR$(251);
4150 NEXT M
4160 NEXT N
4170 RETURN
4180 REM Tecla de funcion 4
         Giro de caracteres
4190 IF M1=2 THEN GOTO 4390
4200 REM Trabajo en el bloque izquierdo
4210 FOR N=1 TO 8
4220 FOR M=1 TO 8
4230 C(N,M)=A(M,9-N)
4240 NEXT M
4250 NEXT N
4260 FOR N=1 TO 8
4270 FOR M=1 TO 8
4280 A(N,M)=C(N,M)
4290 NEXT M
4300 NEXT N
4310 FOR N=0 TO 7
4320 FOR M=0 TO 7
4330 LOCATE N+1,M+12
4340 IF A(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
4350 NEXT M
4360 NEXT N
4370 RETURN
4380 REM Trabajo en el bloque derecho
         (M1=2)
4390 FOR N=1 TO 8
4400 FOR M=1 TO 8
4410 C(N,M)=B(M,9-N)
4420 NEXT M
```

```
4430 NEXT N
4440 FOR N=1 TO 8
4450 FOR M=1 TO 8
4460 B(N,M)=C(N,M)
4470 NEXT M
4480 NEXT N
4490 FOR N=0 TO 7
4500 FOR M=0 TO 7
4510 LOCATE N+10,M+12
4520 IF B(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
4530 NEXT M
4540 NEXT N
4550 RETURN
4560 REM Tecla de funcion 5
        conectar/desconectar pantalla
        auxiliar (cambiar entre pan-
        tallas (VDP(2)) y juegos de
        caracteres (VDP(4))
4570 IF VDP(2)=6 THEN FOR N=6919 TO 6935
 STEP 4:VPOKE N,0:NEXT N:VDP(4)=5:VDP(2)
=9 ELSE VDP(4)=0:VDP(2)=6:FOR N=6919 TO
6935 STEP 4:VPOKE N,15:NEXT N
4580 RETURN
4590 REM Tecla de funcion 6
        grabar juego de caracteres
4600 LOCATE 10,0
4610 OPEN "charac" FOR OUTPUT AS #1
4620 FOR N=33 TO 249
4630 FOR M=0 TO 7
4640 A%=VPEEK(BASE(7)+(N*8)+M)
4650 PRINT #1,A%
4660 NEXT M
4670 NEXT N
4680 CLOSE
4690 RETURN
4700 REM Tecla de funcion 7
        imposible saltar al bloque
```

```
4710 REM derecho, si actualmente se
         encuentra en un bloque editor
          (ED=1)
4720 IF ED=1 THEN RETURN
4730 PUT SPRITE 1,(85,92),15
4740 PUT SPRITE 2,(123,92),15
4750 PUT SPRITE 3,(85,130),15
4760 PUT SPRITE 4,(123,130),15
4770 M1=2
4780 RETURN
4790 REM Tecla de funcion 8
         borrar caracteres
4800 IF M1=2 THEN GOTO 4950
4810 REM Trabajo en el bloque izquierdo
         (M1=1)
4820 FOR N=1 TO 8
4830 FOR M=1 TO 8
4840 A(N,M)=0
4850 NEXT M
4860 NEXT N
4870 FOR N=0 TO 7
4880 FOR M=0 TO 7
4890 LOCATE N+1,M+12
4900 PRINT CHR$(251):
4910 NEXT M
4920 NEXT N
4930 RETURN
4940 REM Trabajo con el bloque derecho
         (M1=2)
4950 FOR N=1 TO 8
4960 FOR M=1 TO 8
4970 B(N,M)=0
4980 NEXT M
4990 NEXT N
5000 FOR N=0 TO 7
5010 FOR M=0 TO 7
5020 LOCATE N+10,M+12
5030 PRINT CHR$(251);
```

```
5040 NEXT M
5050 NEXT N
5060 RETURN
5070 REM Tecla de funcion 9
         Invertir caracteres
5080 IF M1=2 THEN GOTO 5230
5090 REM Trabajo en el bloque izquierdo
         (M1=1)
5100 FOR N=1 TO 8
5110 FOR M=1 TO 8
5120 IF A(N,M)=1 THEN A(N,M)=0 ELSE A(N,
M)=1
5130 NEXT M
5140 NEXT N
5150 FOR N=0 TO 7
5160 FOR M=0 TO 7
5170 LOCATE N+1,M+12
5180 IF A(N+1,M+1)=0 THEN PRINT CHR$(251
); ELSE PRINT CHR$(252);
5190 NEXT M
5200 NEXT N
5210 RETURN
5220 REM Trabajo en el bloque derecho
         (M1=2)
5230 FOR N=1 TO 8
5240 FOR M=1 TO 8
5250 IF B(N,M)=1 THEN B(N,M)=0 ELSE B(N,
M)=1
5260 NEXT M
5270 NEXT N
5280 FOR N=0 TO 7
5290 FOR M=0 TO 7
5300 LOCATE N+10,M+12
5310 IF B(N+1,M+1)=0 THEN PRINT CHR$(251
); ELSE PRINT CHR$(252);
5320 NEXT M
5330 NEXT N
```

```
5340 RETURN
5350 REM Tecla de funcion 10
         intercambiar contenidos de
         bloque 1 y bloque 2
5360 FOR N=1 TO 8
5370 FOR M=1 TO 8
5380 C(N,M)=A(N,M)
5390 A(N,M)=B(N,M)
5400 B(N,M)=C(N,M)
5410 NEXT M
5420 NEXT N
5430 FOR N=0 TO 7
5440 FOR M=0 TO 7
5450 LOCATE N+1,M+12
5460 IF A(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
5470 LOCATE N+10,M+12
5480 IF B(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
5490 NEXT M
5500 NEXT N
5510 RETURN
5520 REM Al interrumpir el programa se
         finaliza todo limpiamente
5530 VDP(2)=6
5540 VDP(4)=0
5550 CLS
5560 SCREEN,3
5570 END
```

Descripción de programa FRETY

Buenos días, querido lector,

El juego, cuyo listado verá a continuación, trata de un pequeño mono que se ha escapado del zoológico y huye ahora del guarda.

A tal fin, al principio debe correr lo más rápidamente posible, puesto que un perseguidor ya le viene pisando los talones.

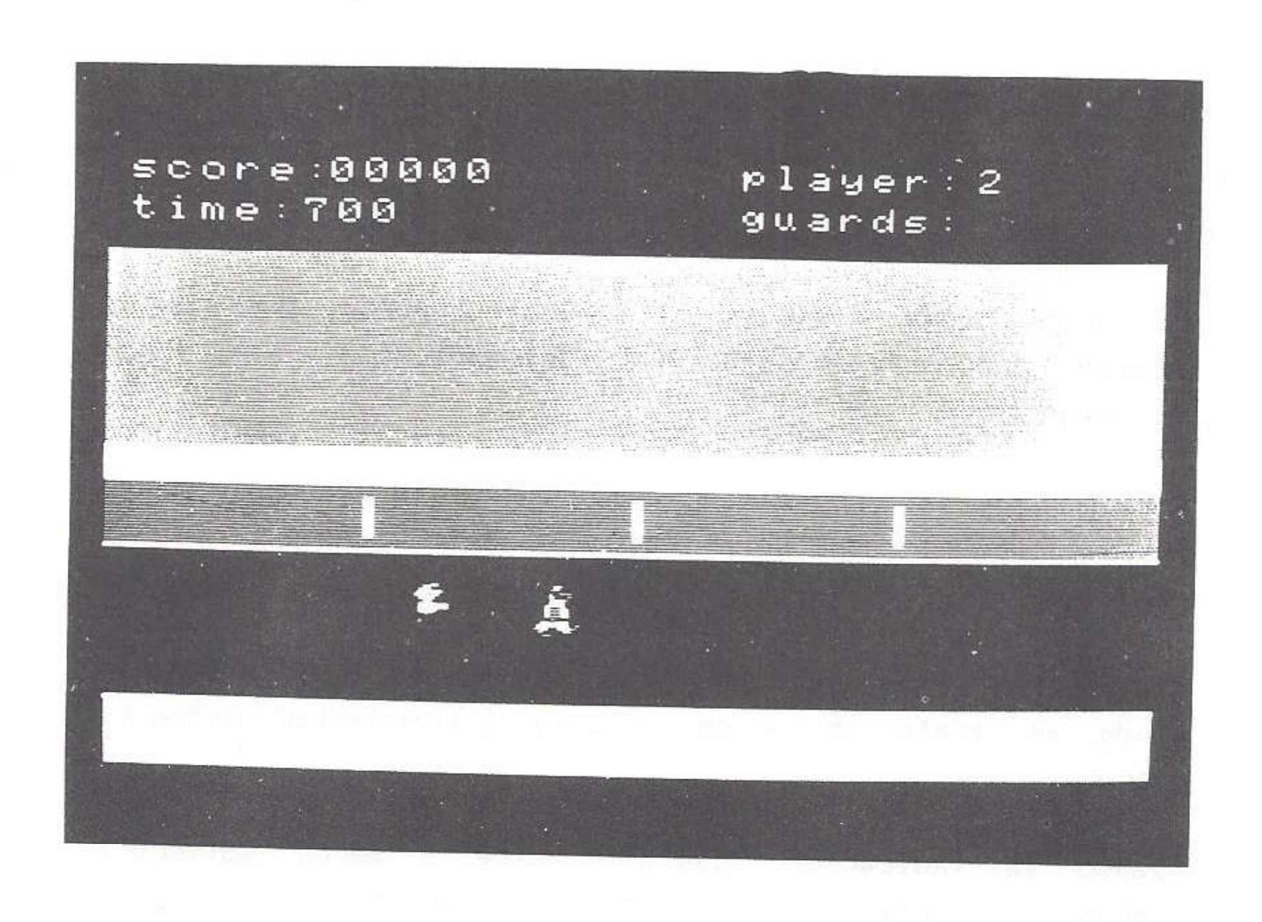
Llegado al final de la calle, aparece la segunda imagen, donde se trata de recoger plátanos y arrojarlos contra el guarda.

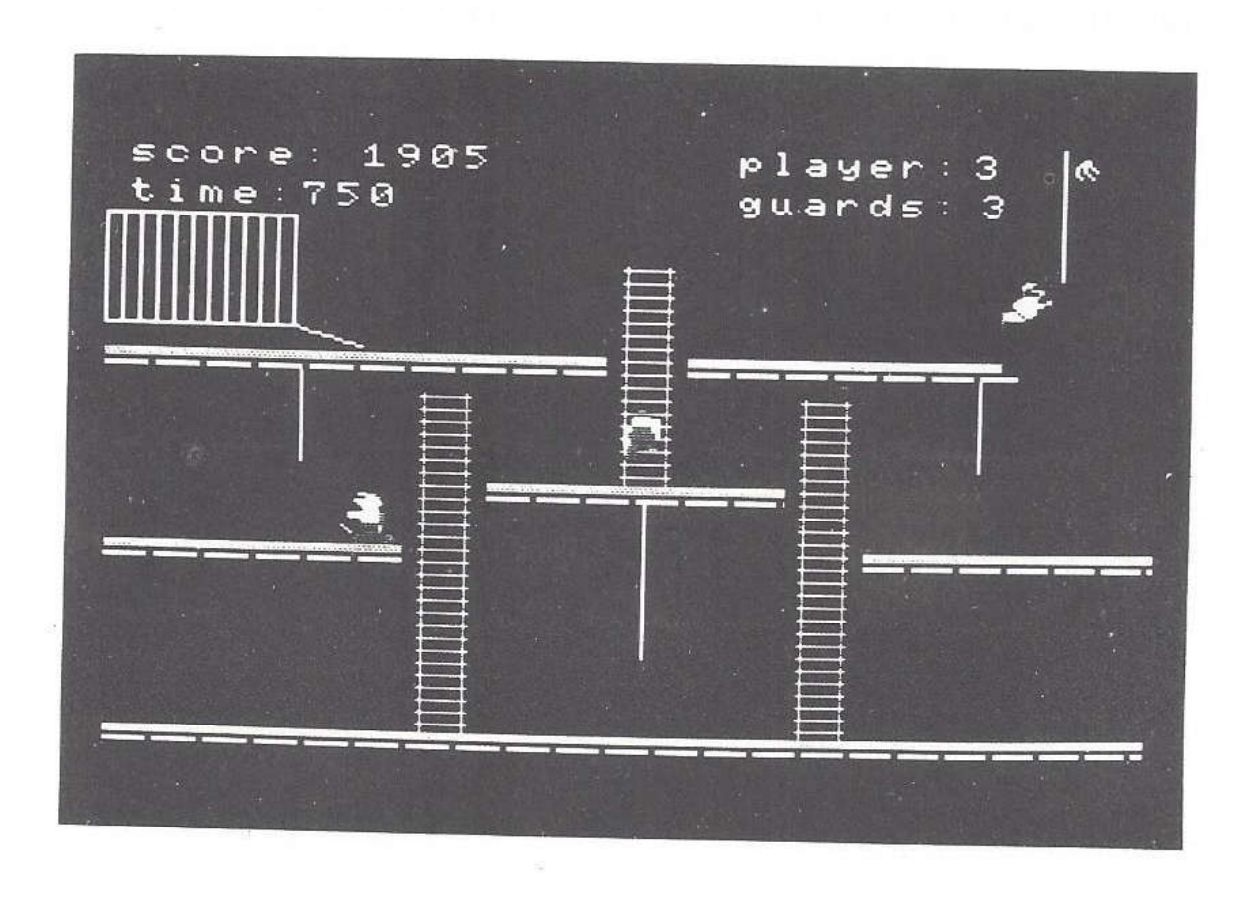
Cuando la indicación "guards:" está a cero, volverá a aparecer la primera imagen donde hay que correr, pero esta vez el perseguidor ha adquirido mayor velocidad.

En cada pasada del juego hay que despistar más guardas, de modo que más tarde la situación puede hacerse crítica cuando haya transcurrido el tiempo.

¡Qué le divierta el juego!

Olaf Otis
(nombre artístico)


score:00000
time:700
player:2
guards:


score: 1905
time:750
player:3
guards: 3

```
10 REM FRETY
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
       Rainer Luers & Olaf Otis
40 DEFINT C-Y,A
50 SCREEN,,0
60 OPEN "grp:"FOR OUTPUT AS #1
70 DIM M(20)
80 DIM SD(8)
90 COLOR 10,1,1
100 SCREEN 2
110 LINE(30,40)-(220,130),15,B
120 DRAW "bm 110,85"
130 PRINT #1,"FRETY"
140 DRAW "bm 28,150"
150 PRINT #1,"DESEEA LEER LAS"
160 DRAW "bm 78,158"
170 PRINT #1,"INSTRUCCIONES?(S/N)"
180 A$=INKEY$
190 IF A$="S" OR A$="s" THEN GOSUB 6920
ELSE IF A$="N" OR A$="n" THEN GOTO 210
200 GOTO 180
210 SCREEN 2
220 DRAW "bm 8,10"
230 PRINT #1,"DESEA JUGAR"
240 DRAW "bm 8,20"
250 PRINT #1,"CON EL TECLADO (0)"
260 DRAW "bm 74,30"
270 PRINT #1,"O CON EL JOYSTICK (1)?"
280 DRAW "bm 2,6"
290 A$=INKEY$
300 IF A$="1" THEN AZ=1:GOTO 330
310 IF A$="0" THEN AZ=0:GOTO 330
320 GOTO 290
330 SCREEN 2,2
340 FOR A=1 TO 30 STEP 8
```

```
350 LINE(30,40)-(220,130),15,B
360 DRAW "bm 48,80"
370 COLOR 10
380 PRINT #1,"AUN TARDARA UN POCO"
390 DRAW "bm 70,90"
400 PRINT #1,"HASTA EL COMIENZO!"
410 COLOR 14
420 FOR L=1 TO 20
430 READ M(L)
440 NEXT L
450 DATA 1,4,7,9,11,12,13,14,15,15,15,15
,14,13,12,11,9,7,4,0
460 FOR CB=1 TO 8
470 READ SD(CB)
480 NEXT CB
490 DATA 5,5,4,4,3,3,4,4
500 CB=0
510 ' Definicion de sprites
520 S$=SPACE$(32)
530 FOR Q=0 TO 13
540 FOR I=1 TO 16
550 READ A$
560 MID$(S$,I,1)=CHR$(VAL("&B"+LEFT$(A$,
8)))
570 MID$(S$,I+16)=CHR$(VAL("&B"+RIGHT$(A
$,8)))
580 NEXT I
590 SPRITE$(Q)=S$
600 NEXT Q
610 RESTORE 1920
620 FOR T=14 TO 25
630 FOR N=1 TO 16
640 READ A$
650 B$=SPACE$(16)
660 FOR U=1 TO LEN(A$)
670 MID$(B$,LEN(A$)+1-U,1)=MID$(A$,U,1)
680 NEXT U
```

```
690 A$=B$
700 MID$(S$,N,1)=CHR$(VAL("&B"+LEFT$(A$,
8)))
710 MID$(S$,N+16)=CHR$(VAL("&B"+RIGHT$(A
$,8)))
720 NEXT N
730 SPRITE$(T)=S$
740 NEXT T
750 ' Indicacion
760 RT=0
770 CV=4
780 ZX=0
790 FA=2
800 ZE=1
810 F$="1"
820 DRAW "bm 52,2"
830 PRINT #1,"00000"
840 DRAW "bm 208,2"
850 PRINT #1,"4"
860 DRAW "bm  6,2"
870 PRINT #1,"score:                  player:"
880 COLOR 1
890 DRAW "bm 80,14"
900 PRINT #1,"    "
910 COLOR 14
920 DRAW "bm  6 ,14"
930 PRINT #1,"time:700                guards:"
940 COLOR 1
950 DRAW "bm 204,14"
960 PRINT #1,"   "
970 COLOR 14
980 DRAW "bm 202,14"
990 PRINT #1,FA
1000 IF F$="1" THEN GOTO 5760 ELSE 1010
1010 ' Pantalla 2
1020 FOR A=0 TO 255 STEP 12
1030 LINE(A,190)-(A+10,191),13,BF
```

```
1040 NEXT A
1050 LINE(0,186)-(255,188),12,BF
1060 FOR A=0 TO 60 STEP 12
1070 LINE(A,130)-(A+10,131),13,BF
1080 LINE(A+90,110)-(A+100,111),13,BF
1090 LINE(A+182,130)-(A+192,131),13,BF
1100 NEXT A
1110 LINE(0,126)-(70,128),12,BF
1120 LINE(90,106)-(162,108),12,BF
1130 LINE(162,110)-(162,111),13
1140 LINE(182,126)-(255,128),12,BF
1150 LINE(254,130)-(255,131),13,BF
1160 FOR A=0 TO 110 STEP 12
1170 LINE(A,70)-(A+10,71),13,BF
1180 LINE(138+A,70)-(148+A,71),13,BF
1190 NEXT A
1200 LINE(220,70)-(255,72),1,BF
1210 LINE(0,66)-(118,68),12,BF
1220 LINE(138,66)-(215,68),12,BF
1230 FOR A=1 TO 108 STEP 4
1240 LINE(74,78+A)-(86,78+A),5
1250 LINE(166,78+A)-(178,78+A),5
1260 NEXT A
1270 FOR A=1 TO 68 STEP 4
1280 LINE(122,38+A)-(134,38+A),5
1290 NEXT A
1300 LINE(123,38)-(123,105),5
1310 LINE(133,38)-(133,105),5
1320 LINE(75,78)-(75,185),5
1330 LINE(85,78)-(85,185),5
1340 LINE(167,78)-(167,185),5
1350 LINE(177,78)-(177,185),5
1360 LINE(128,112)-(128,160),14
1370 LINE(46,72)-(46,100),14
1380 LINE(210,72)-(210,100),14
1390 LINE(230,0)-(230,40),14
1400 FOR A=0 TO 44 STEP 4
```

```
1410 LINE(A,24)-(A,58),14
1420 NEXT A
1430 LINE(0,24)-(44,58),14,B
1440 LINE(44,58)-(60,64),14
1450 PUT SPRITE 5,(230,2),10,13
1460 ' Ejecucion del juego
1470 DRAW"bm186,120f5e5"
1480 X=201
1490 Y=169
1500 U=0
1510 O=60
1520 P=49
1530 A$="R"
1540 Q=4
1550 I=0
1560 L=20
1570 RI=1
1580 C=13
1590 S=2
1600 S1=4
1610 A1=180
1620 I1=0
1630 GOSUB 4770
1640 INTERVAL ON
1650 ON INTERVAL=18 GOSUB 4510
1660 IF I1=0 AND Y=109 AND X<194 AND X>1
74 THEN GOSUB 6670
1670 IF I1=13 AND Y=109 AND X<66 AND X>5
0 THEN GOSUB 6670
1680 IF STICK(AZ)=3 THEN C=0:D=4:GOSUB 4
670
1690 IF STICK(AZ)=7 THEN C=13:D=-4:GOSUB
 4670
1700 IF STICK(AZ)=1 THEN Z=-2:GOSUB 4810
1710 IF STICK(AZ)=5 THEN Z=2:GOSUB 4810
1720 IF POINT(X,Y+17)=12 AND STICK(AZ)=2
 THEN C=0:RI=1:GOSUB 5000
```

```
1730 IF POINT(X,Y+17)=12 AND STICK(AZ)=8
 THEN C=13:RI=-1:GOSUB 5000
1740 IF T$="J" AND STRIG(AZ)=-1 THEN T$=
"N":GOSUB 5280
1750 GOTO 1660
1760 DATA 0000000000000000
1770 DATA 0000000000000000
1780 DATA 0000000000000000
1790 DATA 0000000000000000
1800 DATA 0000000110000000
1810 DATA 0000000110000000
1820 DATA 0000000110000000
1830 DATA 0000000110000000
1840 DATA 0000000110000000
1850 DATA 0000000110000000
1860 DATA 0000000110000000
1870 DATA 0000000110000000
1880 DATA 0000000110000000
1890 DATA 0000000110000000
1900 DATA 0000000110000000
1910 DATA 0000000110000000
1920 '
1930 DATA 0000001110000000
1940 DATA 0000011100000000
1950 DATA 0000010111100000
1960 DATA 0000011001100000
1970 DATA 0000011111000000
1980 DATA 0000000000000000
1990 DATA 0000011100000000
2000 DATA 0000011100000000
2010 DATA 0000011100000000
2020 DATA 0000011100000000
2030 DATA 0000011000000000
2040 DATA 0000111000000000
2050 DATA 0010000000011000
2060 DATA 0111000001110000
2070 DATA 0011000011100000
```

```
2080 DATA 0001110001000000
2090 '
2100 DATA 0000001110000000
2110 DATA 0000011100000000
2120 DATA 0000010111100000
2130 DATA 0000011001100000
2140 DATA 0000011111000000
2150 DATA 0000000000000000
2160 DATA 0000011100000000
2170 DATA 0000011100000000
2180 DATA 0000011100000000
2190 DATA 0000011100000000
2200 DATA 0000011000000000
2210 DATA 0000111000000000
2220 DATA 0000000000000000
2230 DATA 0000000000000000
2240 DATA 0000111000000000
2250 DATA 0000111111000000
2260 '
2270 DATA 0000001100000000
2280 DATA 0000000000000000
2290 DATA 0000000000000000
2300 DATA 0000010000000000
2310 DATA 0000000000000000
2320 DATA 0000011110000000
2330 DATA 0000100010000000
2340 DATA 0000100011000000
2350 DATA 0000100011000000
2360 DATA 0000100011000000
2370 DATA 0000000000000000
2380 DATA 0001100111000000
2390 DATA 0011111111000000
2400 DATA 0001100111000000
2410 DATA 0000000000000000
2420 DATA 0000000000000000
2430 '
2440 DATA 0000001100000000
```

```
2450 DATA 0000000000000000
2460 DATA 0000000000000000
2470 DATA 0000010000000000
2480 DATA 0000000000000000
2490 DATA 0000011110000000
2500 DATA 0000100010000000
2510 DATA 0000100011000000
2520 DATA 0000100011000000
2530 DATA 0000100011000000
2540 DATA 0000000000000000
2550 DATA 0000100111000000
2560 DATA 0000111110000000
2570 DATA 0000111000000000
2580 DATA 0000000000000000
2590 DATA 0000000000000000
2600 '
2610 DATA 0000011111000000
2620 DATA 0000111100000000
2630 DATA 0000111111100000
2640 DATA 0000111111000000
2650 DATA 0000011110000000
2660 DATA 0000000000000000
2670 DATA 0000011111110000
2680 DATA 0000011111110000
2690 DATA 0000011110000000
2700 DATA 0000000000000000
2710 DATA 0000000000000000
2720 DATA 0000000000000000
2730 DATA 0000000000000000
2740 DATA 0000000000000000
2750 DATA 0000000000000000
2760 DATA 0000000000000000
2770 '
2780 DATA 0000011100000000
2790 DATA 0000110000001000
2800 DATA 0000100000001000
2810 DATA 0000001110001000
```

```
2820 DATA 0000000000001000
2830 DATA 0000111110001000
2840 DATA 0001100000001000
2850 DATA 0001100000001000
2860 DATA 0001100001010000
2870 DATA 0001111111000000
2880 DATA 0001111111000000
2890 DATA 0000000000000000
2900 DATA 0010111111000000
2910 DATA 0111011111101000
2920 DATA 1111001111110000
2930 DATA 0011100011100000
2940 '
2950 DATA 0000011100000000
2960 DATA 0000110000000001
2970 DATA 0000100000000001
2980 DATA 0000001110000001
2990 DATA 0000000000000010
3000 DATA 0000111110000010
3010 DATA 0001100000000100
3020 DATA 0001100000001000
3030 DATA 0001100001010000
3040 DATA 0001111111000000
3050 DATA 0001111111000000
3060 DATA 0000000000000000
3070 DATA 0001111111000000
3080 DATA 0001111100000000
3090 DATA 0001110000000000
3100 DATA 0001111100000000
3110 '
3120 DATA 0000000000000000
3130 DATA 0000000000000000
3140 DATA 0000010001000000
3150 DATA 0000011111110000
3160 DATA 0000011111110000
3170 DATA 0000001110110000
3180 DATA 0001100000110000
```

```
3190 DATA 0001100000110000
3200 DATA 0001100000110000
3210 DATA 0001100000000000
3220 DATA 0001100000000000
3230 DATA 0001100000000000
3240 DATA 0000000000000000
3250 DATA 0000000000000000
3260 DATA 0000000000000000
3270 DATA 0000000000000000
3280 '
3290 DATA 0000001110000000
3300 DATA 0000011111000000
3310 DATA 0000011111000000
3320 DATA 0000000000000000
3330 DATA 0000000000000000
3340 DATA 0000011111000000
3350 DATA 0000011111000000
3360 DATA 0000111111100000
3370 DATA 0000111111100000
3380 DATA 0000111111100000
3390 DATA 0000000000000000
3400 DATA 0000111111100000
3410 DATA 0000111011110000
3420 DATA 0000111000000000
3430 DATA 0000111000000000
3440 DATA 0001111000000000
3450 '
3460 DATA 0000000000000000
3470 DATA 0000011111000000
3480 DATA 0000011111110000
3490 DATA 0000011111110000
3500 DATA 0000011111110000
3510 DATA 0000001110110000
3520 DATA 0001100000110000
3530 DATA 0001100000110000
3540 DATA 0001100000110000
3550 DATA 0001100000000000
```

```
3560 DATA 0001100000000000
3570 DATA 0001100011100000
3580 DATA 0001100011110000
3590 DATA 0000000000000000
3600 DATA 0000111000000000
3610 DATA 0001111000000000
3620 '
3630 DATA 0000000011100000
3640 DATA 0000000111000000
3650 DATA 0000000101111000
3660 DATA 0000000100011000
3670 DATA 0000000111110000
3680 DATA 0000000000000110
3690 DATA 0000000111111110
3700 DATA 0000001111111000
3710 DATA 0000000000000000
3720 DATA 0011000000000000
3730 DATA 0011000000000000
3740 DATA 0011000000000000
3750 DATA 0010000000000000
3760 DATA 0010000000000000
3770 DATA 0010000000000000
3780 DATA 0000000000000000
3790 '
3800 DATA 0000000010000000
3810 DATA 0000000100000000
3820 DATA 0000000000000000
3830 DATA 0000000000000000
3840 DATA 0000000000000000
3850 DATA 0000001111100000
3860 DATA 0000011000000000
3870 DATA 0000010000000000
3880 DATA 0000001111100000
3890 DATA 0000110111100000
3900 DATA 0000111011000000
3910 DATA 0000111100000000
3920 DATA 0000000000000000
```

```
3930 DATA 0000000000000000
3940 DATA 0000000000000000
3950 DATA 0000000000000000
3960 '
3970 DATA 0000011000000000
3980 DATA 0000111000000000
3990 DATA 0001010100000000
4000 DATA 0001010010000000
4010 DATA 0001001000000000
4020 DATA 0000100100000000
4030 DATA 0000000000000000
4040 DATA 0000000000000000
4050 DATA 0000000000000000
4060 DATA 0000000000000000
4070 DATA 0000000000000000
4080 DATA 0000000000000000
4090 DATA 0000000000000000
4100 DATA 0000000000000000
4110 DATA 0000000000000000
4120 DATA 0000000000000000
4130 ' guarda izquierda-derecha
4140 GOSUB 5630
4150 O=O+Q
4160 G=G+1
4170 E=(-1)^G
4180 IF E=-1 THEN F=6
4190 IF E=1 THEN F=7
4200 PUT SPRITE 4,(O,P),11,5+I
4210 PUT SPRITE 3,(O,P),4,F+I
4220 IF Q=4 AND POINT(O,P+17)<>12 THEN A
$="T"
4230 IF Q=-4 AND POINT(O+16,P+17)<>12 TH
EN A$="T"
4240 IF O<1 THEN O=60:P=49:I=0:Q=4
4250 IF O>238 THEN O=60:P=49:I=0
4260 IF Y=P AND O>X-8 AND O<X+8 THEN GOS
UB 6670
```

```
4270 RETURN
4280 ' guarda escalera
4290 GOSUB 5630
4300 IF O>130 THEN J=4 ELSE J=0
4310 P=P+2
4320 G=G+1
4330 E=(-1)^G
4340 IF E=-1 THEN K=0
4350 IF E=1 THEN K=13
4360 PUT SPRITE 4,(O+J,P),11,8+K
4370 PUT SPRITE 3,(O+J,P),4,9+K
4380 IF POINT(8+O,P+17)=12 THEN GOSUB 44
20
4390 IF P>Y-16 AND P<Y+16 AND O+J>X-10 A
ND O+J<X+26 THEN GOSUB 6670
4400 RETURN
4410 ' direccion
4420 A$="R"
4430 IF X<128 AND P<120 THEN Q=-4:I=13
4440 IF X>128 AND P<120 THEN Q=4:I=0
4450 IF X>160 AND P>120 THEN Q=4:I=0
4460 IF X<80 AND P>120 THEN Q=-4:I=13
4470 IF X<160 AND O>150 AND P>120 THEN Q
=-4:I=13
4480 IF X>80 AND O<90 AND P>120 THEN Q=4
:I=0
4490 RETURN
4500 ' intervalo
4510 ZX=ZX+1
4520 IF ZX MOD 50=0 THEN LINE(46,14)-(66
,22),1,BF:DRAW "bm 46,14":COLOR 14:PRINT
 #1,USING "###";900-ZX/2
4530 IF ZX=1800 THEN GOSUB 6680
4540 CB=CB+1
4550 IF CB=9 THEN CB=1
4560 SOUND 8,16
4570 SOUND 12,6
```

```
4580 SOUND 13,9
4590 SOUND 1,SD(CB)
4600 SOUND 10,16
4610 SOUND 4,2
4620 SOUND 12,5
4630 SOUND 13,8
4640 IF A$="R" THEN GOSUB 4140
4650 IF A$="T" THEN GOSUB 4290
4660 RETURN
4670 ' mono izquierda-derecha
4680 X=X+D
4690 IF X>240 THEN X=X-4
4700 IF X<2 THEN X=X+4
4710 IF POINT(X-5,Y+17)<>12 AND POINT(X+
24,Y+17)<>12 THEN GOSUB 4920
4720 U=U+1
4730 V=(-1)^U
4740 IF V=-1 THEN S=1
4750 IF V=1 THEN S=2
4760 IF STICK(AZ)<>3 AND STICK(AZ)<>7 TH
EN S=2
4770 PUT SPRITE 2,(X,Y),6,S+C
4780 PUT SPRITE 1,(X,Y),10,2+S+C
4790 RETURN
4800 ' mono escalera
4810 IF POINT(X+2,Y)=5 AND POINT(X+12,Y)
=5 THEN GOTO 4820 ELSE RETURN
4820 IF Z=2 AND POINT(X+8,Y+17)=12 THEN
RETURN
4830 Y=Y+Z
4840 U=U+1
4850 V=(-1)^U
4860 IF V=-1 THEN S=0
4870 IF V=1 THEN S=13
4880 PUT SPRITE 2,(X,Y),10,9+S
4890 PUT SPRITE 1,(X,Y),6,10+S
4900 RETURN
```

```
4910 ' caida
4920 IF STICK(AZ)=7 THEN X=X-8
4930 IF STICK(AZ)=3 THEN X=X+8
4940 Y=Y+1
4950 PUT SPRITE 2,(X,Y),10,9
4960 PUT SPRITE 1,(X,Y),6,10
4970 IF POINT(X,Y+17)=12 OR POINT(X+16,Y
+17)=12 GOTO 1660
4980 GOTO 4940
4990 ' salto
5000 FOR L=1 TO 20
5010 PUT SPRITE 1,(X+RI*L,Y-M(L)),10,12+
C
5020 PUT SPRITE 2,(X+RI*L,Y-M(L)),6,11+C
5030 IF POINT(X+RI*L+8,Y-M(L))=14 THEN G
OSUB 5100
5040 IF X+RI*L+8>O AND X+RI*L+8<O+16 AND
 Y-M(L)+16>P AND Y-M(L)+16<P+16 THEN GOT
O 6680
5050 NEXT L
5060 X=X+RI*20
5070 D=0
5080 GOTO 4670
5090 '
5100 Y=Y-8
5110 PUT SPRITE 2,(X+RI*L,Y-M(L)),10,3+C
5120 PUT SPRITE 1,(X+RI*L,Y-M(L)),6,1+C
5130 IF STICK(AZ)=5 THEN GOSUB 5190
5140 IF STICK(AZ)=1 THEN Y=Y-1
5150 IF POINT(X+RI*L+8,Y-M(L))<>14 THEN
Y=Y+1
5160 IF Y-M(L)<5 THEN BEEP:PUT SPRITE 5,
(200,200):T$="J":GOSUB 5190
5170 GOTO 5110
5180 '
5190 Y=Y+1
5200 PUT SPRITE 1,(X+RI*L,Y-M(L)),10,12+
C
```

```
5210 PUT SPRITE 2,(X+RI*L,Y-M(L)),6,11+C
5220 IF POINT(X+RI*L,Y-M(L)+17)=12 THEN
X=X+RI*8:Y=Y-M(L):GOTO 5240
5230 GOTO 5190
5240 PUT SPRITE 2,(X,Y),6,2+C
5250 PUT SPRITE 1,(X,Y),10,4+C
5260 GOTO 1660
5270 ' lanzamiento
5280 IF C=0 THEN AS=6
5290 IF C=13 THEN AS=-6
5300 PUT SPRITE 1,(X,Y),10,12+C
5310 PUT SPRITE 2,(X,Y),6,11+C
5320 FOR FG=1 TO 20
5330 PUT SPRITE 5,(X+FG*AS+8,Y-5),10,13
5340 IF X+FG*AS+8>O AND X+FG*AS+8<O+16 A
ND Y=P THEN V$="S":GOSUB 5440
5350 IF X+FG*AS+8>A1 AND X+FG*AS+8<A1+16
 AND Y=109 THEN V$="L":GOSUB 5440
5360 IF X+FG*AS+8>250 THEN 5390
5370 IF X+FG*AS+8<2 THEN 5390
5380 NEXT FG
5390 PUT SPRITE 2,(X,Y),6,5+C
5400 PUT SPRITE 1,(X,Y),10,2+5+C
5410 PUT SPRITE 5,(230,2),10,13
5420 GOTO 1750
5430 '
5440 RT=RT+1
5450 ER=ER+3300
5460 O=60
5470 P=49
5480 BEEP
5490 IF V$="S" THEN O=60:P=49:Q=4:I=0
5500 IF V$="L" THEN A1=180:PUT SPRITE 5,
(100,200):PUT SPRITE 6,(100,200)
5510 COLOR 1
5520 DRAW "bm 204,14"
```

```
5530 LINE(204,14)-(220,22),1,BF
5540 IF FA-RT=0 THEN ER=ER+1000:INTERVAL
 OFF:ZX=0:RT=0:ZE=ZE+.1:GOTO 5760
5550 LINE(50,2)-(100,10),1,BF
5560 COLOR 14:DRAW "bm 196,14":PRINT #1,
FA-RT
5570 LINE(50,2)-(100,10),1,BF
5580 DRAW "bm 52,2"
5590 PRINT #1,USING "#####";ER
5600 IF ERR>10000 THEN ER=0
5610 GOTO 5390
5620 ' guarda 2
5630 A1=A1+S1
5640 Q1=Q1+1
5650 W1=(-1)^Q1
5660 IF W1=-1 THEN F1=6
5670 IF W1=1 THEN F1=7
5680 PUT SPRITE 7,(A1,109),11,5+I1
5690 PUT SPRITE 6,(A1,109),4,F1+I1
5700 IF A1>240 THEN A1=6
5710 IF A1>170 AND A1<180 THEN S1=4:I1=0
:DRAW "c14bm186,120f5e5":DRAW "c1bm56,12
0f5e5"
5720 IF A1>60 AND A1<70 THEN S1=-4:I1=13
:DRAW"c14bm56,120f5e5":DRAW "c1bm186,120
f5e5"
5730 IF A1<2 THEN A1=236:S1=-4:I1=13
5740 IF A1>X-8 AND Y=109 AND A1<8+X THEN
 GOSUB 6670
5750 RETURN
5760 ' Pantalla 1
5770 FOR A=1 TO 7
5780 PUT SPRITE A,(100,200)
5790 NEXT A
5800 LINE(0,40)-(250,190),1,BF
5810 COLOR 1
5820 LINE(204,14)-(220,22),1,BF
```

```
5830 COLOR 14
5840 LINE(0,24)-(60,40),1
5850 LINE(230,0)-(230,40),1
5860 LINE(0,24)-(44,40),1,BF
5870 LINE(0,120)-(256,170),1,BF
5880 LINE(0,30)-(255,90),5,BF
5890 LINE(0,101)-(255,120),12,BF
5900 LINE(0,90)-(255,100),10,BF
5910 LINE(0,170)-(255,191),2,BF
5920 LINE(0,121)-(255,121),15
5930 LINE(0,169)-(255,169),15
5940 B=0:K$="N":GOSUB 6110
5950 INTERVAL ON
5960 ON INTERVAL=12 GOSUB 6220
5970 Z=0
5980 IF K$="N" AND STICK(AZ)=3 THEN K$="
J":GOSUB 6010
5990 IF K$="J" AND STICK(AZ)=7 THEN K$="
N":GOSUB 6010
6000 GOTO 5980
6010 SOUND 9,16
6020 SOUND 13,9
6030 SOUND 12,3
6040 SOUND 3,0
6050 SOUND 7,B4
6060 U=U+1
6070 E=(-1)^U
6080 IF E=-1 THEN S=0
6090 IF E=1 THEN S=1
6100 Z=Z+.5
6110 PUT SPRITE 2,(100+Z,130),6,1+S
6120 PUT SPRITE 1,(100+Z,130),10,3+S
6130 A=A-4
6140 IF A=64 THEN A=0
6150 PUT SPRITE 5,(A+50,100),15,0
6160 PUT SPRITE 6,(A+114,100),15,0
6170 PUT SPRITE 7,(A+178,100),15,0
```

```
6180 PUT SPRITE 8,(A+246,100),15,0
6190 IF 100+Z>240 THEN GOTO 6320
6200 RETURN
6210 ' guarda
6220 RG=RG+1
6230 T=(-1)^RG
6240 IF T=-1 THEN I=6
6250 IF T=1 THEN I=7
6260 B=B+ZE
6270 PUT SPRITE 4,(50+B,130),11,5
6280 PUT SPRITE 3,(50+B,130),4,I
6290 IF 64+B>100+Z THEN GOTO 6540
6300 RETURN
6310 ' Fin pantalla 1
6320 INTERVAL OFF
6330 LINE(0,30)-(255,191),1,BF
6340 COLOR 14,1,1
6350 PUT SPRITE 1,(100,200)
6360 PUT SPRITE 2,(100,200)
6370 PUT SPRITE 3,(100,200)
6380 PUT SPRITE 4,(100,200)
6390 PUT SPRITE 5,(100,200)
6400 PUT SPRITE 6,(100,200)
6410 PUT SPRITE 7,(100,200)
6420 PUT SPRITE 8,(100,200)
6430 ER=ER+(100+Z-50+O)*10
6440 BEEP
6450 COLOR 1
6460 LINE(50,2)-(100,10),1,BF
6470 COLOR 14
6480 DRAW "bm 52,2"
6490 PRINT #1,USING "#####";ER
6500 IF ER>10000 THEN ER=0
6510 FA=FA+1
6520 F$="2"
6530 GOTO 860
6540 ' Retirada mono 1
```

```
6550 Z=0
6560 B=0
6570 CV=CV-1
6580 BEEP
6590 LINE(201,2)-(220,10),1,BF
6600 COLOR 14
6610 DRAW"BM 200,2"
6620 PRINT #1,CV
6630 IF CV=0 THEN GOSUB 6800
6640 FOR R=1 TO 500
6650 NEXT R
6660 RETURN
6670 ' Retirada mono 2
6680 CV=CV-1
6690 BEEP
6700 LINE(201,2)-(220,10),1,BF
6710 COLOR 14
6720 DRAW "bm 200,2"
6730 PRINT #1,CV
6740 IF CV=0 THEN GOSUB 6800
6750 FOR R=1 TO 500
6760 NEXT R
6770 PUT SPRITE 5,(230,2),10,13
6780 GOTO 1480
6790 ' Fin juego
6800 INTERVAL OFF
6810 SOUND 8,0
6820 SOUND 9,0
6830 SOUND 10,0
6840 LINE(85,80)-(168,110),4,BF
6850 COLOR 15
6860 DRAW "bm 92,90"
6870 PRINT #1,"GAME OVER"
6880 DRAW "bm 92,98"
6890 PRINT #1,"continuar->s"
6900 A$=INKEY$
6910 IF A$="s" OR A$="S" THEN RUN ELSE G
OTO 6900
```

```
6920 SCREEN 2
6930 DRAW "bm 12,10"
6940 PRINT #1,"Pantalla 1:moviendo el jo
y-"
6950 DRAW "bm 68,18"
6960 PRINT #1,"stick (o bien el tec-"
6970 DRAW "bm 68,26"
6980 PRINT #1,"lado), el mono avanza"
6990 DRAW "bm 68,34"
7000 PRINT #1,"para huir del guarda"
7010 DRAW "bm 12,50"
7020 PRINT #1,"Pantalla 2:el objetivo de
"
7030 DRAW "bm 68,58"
7040 PRINT #1,"esta pantalla consiste"
7050 DRAW "bm 68,66"
7060 PRINT #1,"en recoger platanos y"
7070 DRAW "bm 68,74"
7080 PRINT #1,"lanzarlos sobre el"
7090 DRAW "bm 68,82"
7100 PRINT #1,"guarda."
7110 DRAW "bm 68,98"
7120 PRINT #1,"Para saltar, deben"
7130 DRAW "bm 68,106"
7140 PRINT #1,"pulsarse simultaneamen-"
7150 DRAW "bm 68,114"
7160 PRINT #1,"te 2 teclas del cursor"
7170 DRAW "bm 68,122"
7180 PRINT #1,"(p.e.arriba y derecha,"
7190 DRAW "bm 68,130"
7200 PRINT #1,"o sea, salto hacia"
7210 DRAW "bm 68,138"
7220 PRINT #1,"arriba a la derecha)."
7230 DRAW "bm 12,170"
7240 PRINT #1,"Pulse la barra espaciador
a!"
7250 IF STRIG(0)=-1 THEN GOTO 210 ELSE 7
250
```

Descripción de programa MISTER MINER

Apreciados lectores,

Delante de ustedes está el principio de un largo listado y, puesto que los demás tampoco son mucho más cortos, supongo que todavía no ha tecleado ninguno de ellos. Así explicaré brevemente nuestro trabajo:

Nosotros confeccionamos este juego con un SPECTRAVIDEO-328. Puesto que éste no es un verdadero ordenador MSX -como seguramente lo será el suyo- el señor Lüers, autor de este libro, debió efectuar algunos cambios en el juego y probarlo después con un ordenador MSX. Utilizó un aparato de dictar para teclear el programa. Puedo recomendarle esta buena idea, a menos que tenga alguien que le vaya dictando, porque se liará máximo al llegar a los DATAs y comandos DRAW. Es aconsejable dejar el ordenador conectado hasta llegar a la última línea del listado. El ordenador no quedará dañado por estar cierto tiempo en funcionamiento. Si al final del trabajo hay algo que falla, compruebe su listado con el original del libro.

Respecto al juego: la figura principal representa un pequeño minero, por lo cual se llama MR MINER. Se mueve a izquierda y derecha, y en la escalera también arriba y abajo, puede saltar e incluso disparar si ha recargado en la bomba de agua que se encuentra abajo a la derecha.

Las funciones son las habituales, pero para saltar hay que mover el joystick en dirección diagonal hacia arriba y derecha o izquierda; o pulsar las correspondientes teclas del cursor "arriba" e "izquierda" o "arriba" y "derecha". El botón de fuego o la barra espaciadora sirven para disparar.

Puede hacerse una idea a través de la foto. Además existe un murciélago, que lanza un huevo que se convierte en un diamante al llegar al suelo. Por cada diamante obtiene 1000 puntos, y 2000 por el de la izquierda abajo. Para recogerlos hay que dirigir a MR MINER exactamente encima de un diamante

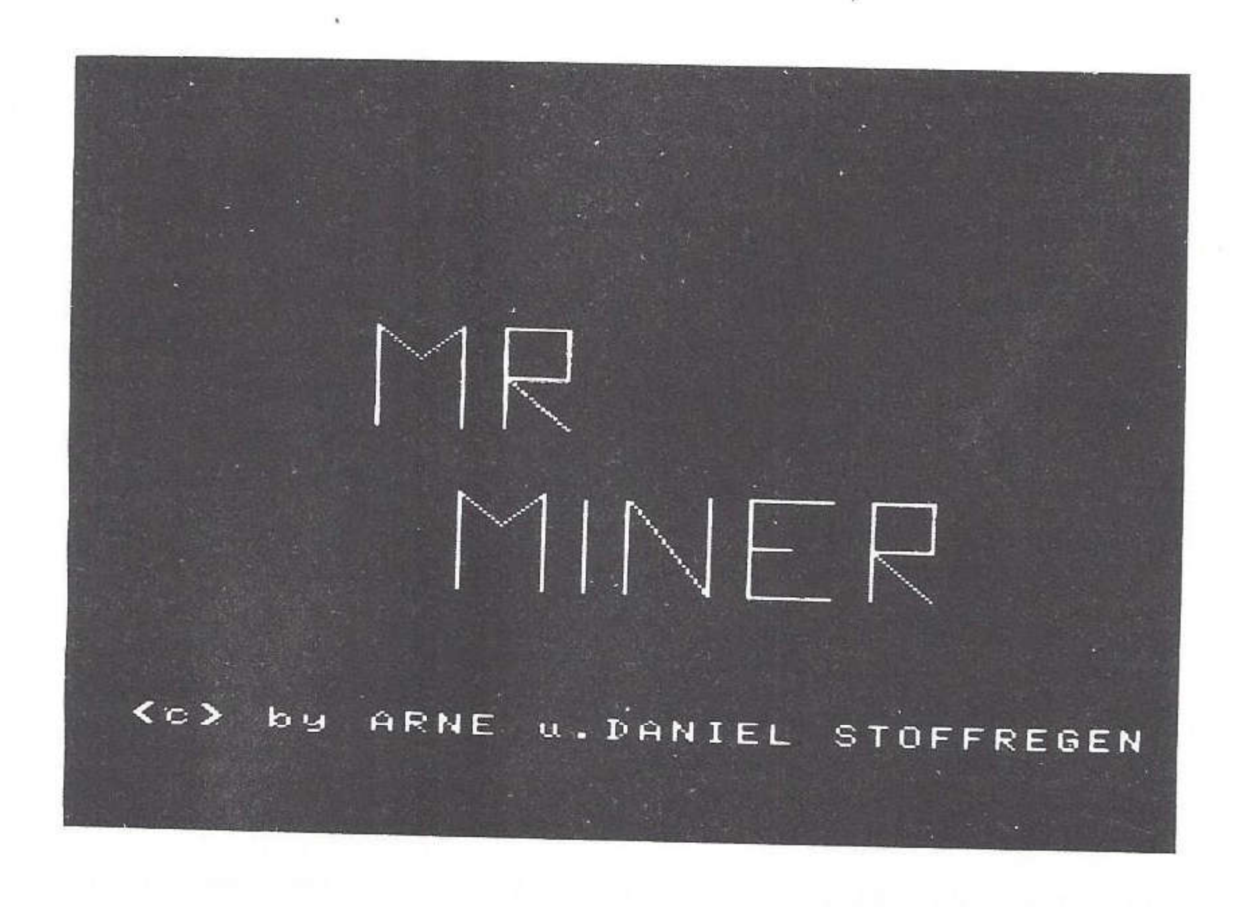
y bajar la palanca. Sin embargo, este diamante influye muy poco en su tiempo disponible, que comienza a contar desde 5000, y al llegar a 0 perderá un MR MINER. La única forma de alargar su vida consiste en recoger diamantes, cada uno de los cuales incrementa el tiempo en 1000 puntos.

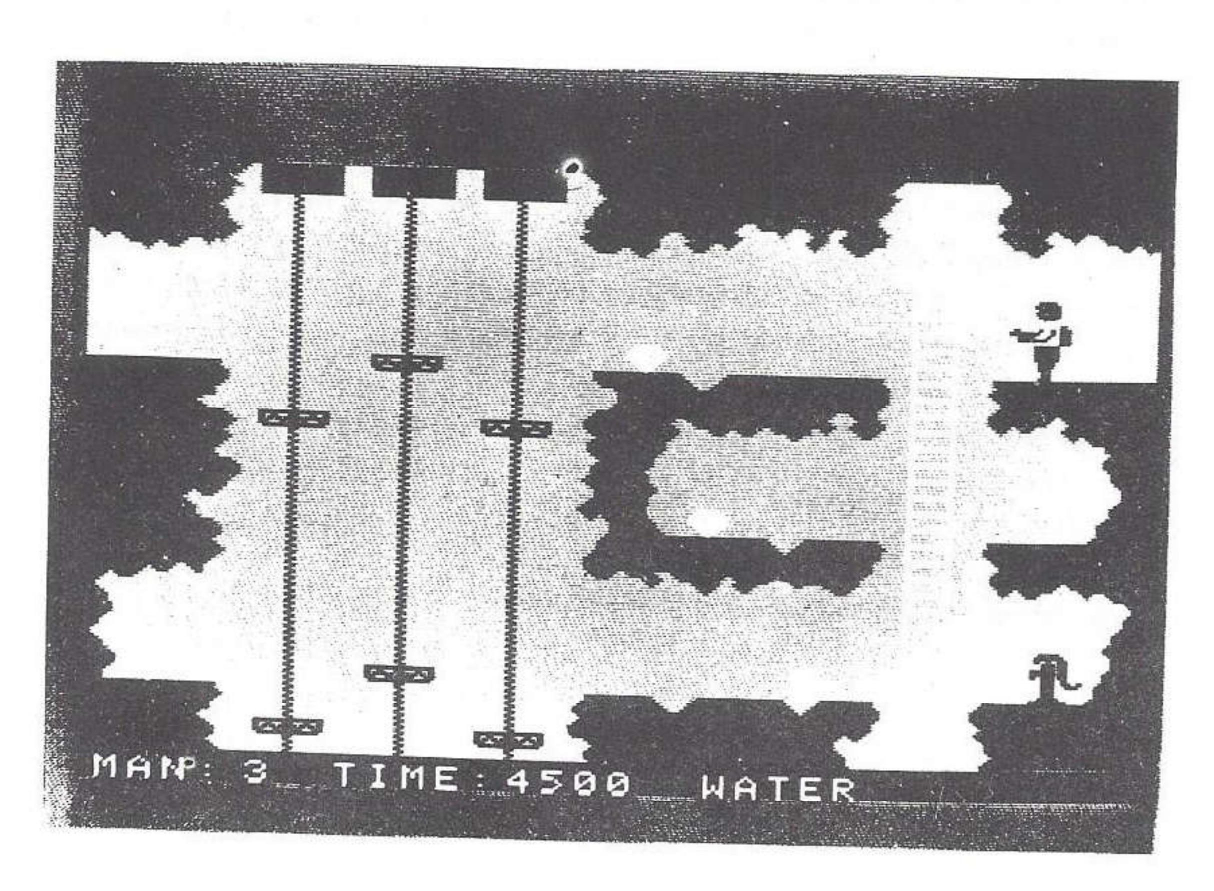
Bien, usted comienza el juego en la esquina superior izquierda; para poder recargar en la bomba de agua abajo a la derecha, pulse simplemente el botón de fuego. Abajo en "WATER" se llenará una franja de color azul. Cada disparo provoca una disminución de su longitud, por lo cual precisa recargar varias veces a lo largo de la partida.

Cuando el depósito en su espalda está cargado, MR MINER podrá utilizar su pistola de agua. Sin embargo, ésta sólo es peligrosa para un fantasma, cuyo contacto debe evitarse. Cada fantasma da 100 puntos.

Si cree que el pasillo arriba a la derecha conduce a otra imagen, debo decepcionarle. También el sonido, si existe, deja mucho que desear. Los problemas que hallamos a la hora de programar el juego no sólo fueron los de encontrarse demasiados sprites en un línea -que provocan un borrado momentáneo- o del lenguaje BASIC un poco lento, sino del poco tiempo de que dispusimos para confeccionar el programa. Naturalmente habíamos planeado más cosas, pero esto no tiene que retenerle a ampliar el programa o de crear una segunda imagen, gracias a sus conocimientos adquiridos de programación. También puede mejorar, dificultar o modificar el programa. Su fantasía no tiene límites y le deseamos que se divierta.

Los programadores: Arne y Daniel Stoffregen


MR
MINER
<c> by ARNE u.DANIEL STOFFREGEN


MAP: 3 TIME:4500 WATER

```
10 REM MISTER MINER
20 REM MSX Graficos y sonido
30 REM Copyright 1985 DATA BECKER &
       Rainer Luers & Arne y
       Daniel Stoffregen
40 OPEN "grp:"FOR OUTPUT AS #1
50 GOSUB 6820
60 CLEAR 4000
70 OPEN "grp:" FOR OUTPUT AS #1
80 DEFINT A-Z
90 ON STRIG GOSUB 6360,6360,6360
100 DIM SP$(35),B(5),C(5),H$(20),H(20)
110 DATA 5,7,6,7,15,17,16,17
120 REM Leer posiciones de movimiento
130 FOR T=0 TO 7
140 READ F(T)
150 NEXT T
160 VFL=5
170 GOSUB 6110
180 DRAW "a0"
190 REM Operadores de salto (+9,-4 etc.)
200 DATA +9,-4,+9,-2,+9,+2
210 RESTORE 200
220 FOR T=1 TO 3
230 READ B(T),C(T)
240 NEXT T
250 SCREEN 2,2
260 GOSUB 2120 'Sprites
270 FOR T=1 TO 31
280 SPRITE$(T)=SP$(T)
290 NEXT T
300 Q$="r15d311d116d111u116u111u3d3r1e2f
2e2r1d1r1f1e2f2d116u211d2"
310 DATA 9,10,7,8,3,4
320 REM Leer sonido
330 RESTORE 310
340 FOR T=1 TO 6
```

```
350 READ S(T)
360 NEXT T
370 REM Definir tiempo para distancia
380 ON INTERVAL=100 GOSUB 6790
390 REM Activar todas las variables
        al valor inicial
400 A3=160
410 A5=96
420 A4=106
430 A6=76
440 M=4
450 O=194
460 P=-1
470 F=240
480 REM Coordenadas iniciales de la
        figura de juego
490 X=0
500 Y=20
510 N=0
520 A1=10
530 A2=0
540 L=1
550 L1=0
560 BQ=0
570 WA$="LEER"
580 LINE (190,182)-(256,192),1,BF
590 ZEIT=5000
600 MN=INT(RND(-TIME)*5)+1
610 EI$=""
620 OA=25
630 PUT SPRITE 29,(0,0),0,31
640 X10=240
650 LE$=""
660 REM Definir sonido
670 SOUND 7,&B00110100
680 SOUND 1,13
690 SOUND 12,2
```

```
700 SOUND 11,55
710 SOUND 8,16
720 SOUND 6,245
730 SOUND 3,6
740 SOUND 9,8
750 IF LI$="N" THEN LI$="":LINE (39,9)-(
109,180),3,BF:JJK$="J":GOSUB 5410
760 PUT SPRITE 27,(0,0),0,31
770 STRIG(1) ON
780 STRIG(0) ON
790 REM Conectar intervalo
800 INTERVAL ON
810 REM Hacer aparecer los diamantes
        en sus diversas posiciones
820 DRAW "bm15,159;"+DI$
830 P$(1)="J"
840 DRAW "bm146,109;"+DI$
850 P$(3)="J"
860 DRAW "BM130,59;"+DI$
870 P$(5)="J"
880 REM Comienzo programa principal
890 A=STICK(0)
900 IF A=0 THEN GOSUB 1560 ELSE IF FA$="
N" THEN ON A GOSUB 1670,1440,1450,1430,1
800,1430,1500,1550
910 REM Direccion (hacia derecha
        o izquierda)
920 IF R$="1" THEN I=9 ELSE I=0
930 IF J$<>"" THEN 950 ELSE GOSUB 1990
940 REM Deben parar los ascensores?
950 IF LI$="N" THEN A2=A2-4:A5=A5-4:GOTO
 1020 ELSE A1=A1+2:A2=A2-4:A3=A3+3:A4=A4
+2:A5=A5-4:A6=A6+3
960 IF A1>192-12 THEN A1=0 ELSE IF A2<0
THEN A2=192-12 ELSE IF A3>192-12 THEN A3
=0 ELSE IF A4>192-12 THEN A4=0 ELSE IF A
5<0 THEN A5=192-12 ELSE IF A6>192-12 THE
N A6=0
```

```
970 REM Activar los ascensores
980 PUT SPRITE 10,(40,A1),1,24
990 PUT SPRITE 12,(92,A3),1,24
1000 PUT SPRITE 13,(40,A4),1,24
1010 PUT SPRITE 15,(92,A6),1,24
1020 IF LI$="N" THEN PUT SPRITE 27,(X10,
22),1,ER ELSE 1130
1030 REM Hacer volar pajaro de derecha
        a izquierda
1040 X10=X10-5
1050 IF X10<=0 THEN X10=240:OA=25
1060 REM Mover alas
1070 RE=RE+1
1080 IF RE/4<>INT(RE/4) THEN IF ER=25 TH
EN ER=26 ELSE ER=25
1090 IF EI$="J" THEN 1100 ELSE IF X10<PX
(MN) AND X10+20>PX(MN) AND P$(MN)="N" TH
EN EI$="J" ELSE MN=INT(RND(-TIME)*5)+1:G
OTO 1130
1100 PUT SPRITE 29,(PX(MN),OA),15,27
1110 OA=OA+5
1120 IF OA>=PY(MN) THEN EI$="":FG$=STR$(
PX(MN)):PUT SPRITE 29,(0,0),0,31:GF$=STR
$(PY(MN)):DRAW "bm"+FG$+","+GF$+DI$:P$(M
N)="J"
1130 PUT SPRITE 11,(66,A2),1,24
1140 PUT SPRITE 14,(66,A5),1,24
1150 IF LI$="N" THEN IF A2<0 THEN A2=180
1160 IF LI$="N" THEN IF A5<0 THEN A5=180
1170 PUT SPRITE 25,(O,P),15,M
1180 IF X>30 AND X+9<56 THEN GOSUB 2050
1190 IF X>56 AND X+9<82 THEN GOSUB 2070
1200 IF X>82 AND X+9<108 THEN GOSUB 2090
1210 REM Activar color al
        saltar escalera
```

```
1220 IF X>185 AND X+16<222 THEN V=11 ELS
E V=12
1230 REM Mover fantasma
1240 IF H$(L)="L" THEN O=O-10 ELSE IF H$
(L)="R" THEN O=O+10 ELSE IF H$(L)="O" TH
EN P=P-10 ELSE IF H$(L)="U" THEN P=P+10
1250 L1=L1+1
1260 IF L1>H(L) THEN L1=1:L=L+1:IF H$(L)
="E" THEN O=194:P=0:L=1:L1=0
1270 IF M=4 THEN M=14 ELSE M=4
1280 REM Salto hacia la derecha
1290 IF J$="R" THEN X=X+B(J1):Y=Y+C(J1):
J1=J1+1:IF J1=3 THEN J$="":J1=0
1300 IF SCHLUSS=1 THEN GOSUB 6380
1310 REM Salto hacia la izquierda
1320 IF J$="L" THEN X=X-B(J2):Y=Y+C(J2):
J2=J2+1:IF J2=3 THEN J$="":J2=0
1330 IF O>=X-4 AND O<=X+6 AND P>=Y-4 AND
 P<=Y+17 THEN 6270
1340 REM PUTSPRITE para salto hombrecito
1350 GOSUB 1580
1360 REM Ascensores paran si/no
1370 BQ=BQ+1
1380 IF BQ=209 THEN GOSUB 6520
1390 DF=DF+1
1400 IF DF=7 THEN DF=1
1410 SOUND 3,S(DF)
1420 GOTO 890
1430 RETURN
1440 IF LE$="J" OR JP$="J" THEN RETURN E
LSE J$="R":R$="r":JP$="J":RETURN
1450 IF LE$="J" THEN LE$="":U=7:R$="r":R
ETURN ELSE SOUND 13,9:R=R+1:IF R=4 THEN
R=0
1460 U=F(R)
1470 X=X+4
```

```
1480 R$="r"
1490 RETURN
1500 IF LE$="J" THEN LE$="":U=16:R$="l":
RETURN ELSE SOUND 13,9:K=K+1:IF K=4 THEN
 K=0
1510 U=F(K+4)
1520 X=X-4
1530 R$="l"
1540 RETURN
1550 IF LE$="J" OR JP$="J" THEN RETURN E
LSE J$="L":R$="l":JP$="J":RETURN
1560 IF R$="l" THEN U=17 ELSE U=7:RETURN
1570 RETURN
1580 IF R$="l" THEN I=10 ELSE I=0
1590 REM Determinar direccion
        al saltar
1600 IF J$="R" THEN U=8 ELSE IF J$="L" T
HEN U=18
1610 IF LE$="J" THEN RETURN
1620 PUT SPRITE 0,(X,Y),1,1+I
1630 PUT SPRITE 1,(X,Y),8,3+I
1640 PUT SPRITE 2,(X,Y+16),4,U
1650 PUT SPRITE 3,(X,Y),7,2+I:RETURN
1660 REM Subir escalera
1670 IF X<191 OR X>205 THEN RETURN
1680 REM Representacion por detras
1690 Y=Y-5
1700 X=192
1710 PUT SPRITE 0,(X,Y),1,21
1720 PUT SPRITE 1,(X,Y),8,22
1730 PUT SPRITE 2,(X,Y),7,23
1740 REM Mover piernas
1750 G=G+1
1760 IF G/2<>INT(G/2) THEN G1=9 ELSE G1=
19:G=0
1770 PUT SPRITE 3,(X,Y+16),4,G1
1780 LE$="J"
```

```
1790 RETURN
1800 REM Bajar escalera o recoger
         diamante
1810 IF X>191 AND X<205 THEN 1890
1820 IF X>5 AND X<12 AND Y=133 AND P$(1)
="J" THEN LINE(11,140)-(20,159),3,BF:P$(
1)="N":ZEI=ZEI+2000:RETURN
1830 IF X>115 AND X<123 AND Y=133 AND P$
(2)="J" THEN LINE(121,140)-(130,159),3,B
F:P$(2)="N":ZEI=ZEI+1000:RETURN
1840 IF X>136 AND X<143 AND Y=83 AND P$(
3)="J" THEN LINE(142,90)-(151,109),3,BF:
P$(3)="N":ZEI=ZEI+1000:RETURN
1850 IF X>220 AND X<227 AND Y=83 AND P$(
4)="J" THEN LINE(226,90)-(235,109),3,BF:
P$(4)="N":ZEI=ZEI+1000:RETURN
1860 IF X>120 AND X<127 AND Y=33 AND P$(
5)="J" THEN LINE(126,40)-(135,59),3,BF:P
$(5)="N":ZEI=ZEI+1000:RETURN
1870 RETURN
1880 REM Representacion por detras
1890 Y=Y+5
1900 X=192
1910 PUT SPRITE 0,(X,Y),1,21
1920 PUT SPRITE 1,(X,Y),8,22
1930 PUT SPRITE 2,(X,Y),7,23
1940 G=G+1
1950 IF G/2<>INT(G/2) THEN G1=9 ELSE G1=
19:G=0
1960 PUT SPRITE 3,(X,Y+16),4,G1
1970 LE$="J"
1980 RETURN
1990 IF POINT(X+8,Y+27)=V THEN FA$="N":N
=0:JP$="":RETURN ELSE FA$="J"
2000 FOR T=1 TO 5
2010 Y=Y+1
2020 IF POINT(X+8,Y+27)<>V THEN NEXT T E
LSE T=6:FA$="N"
```

```
2030 IF N=11 THEN 6270 ELSE N=N+1
2040 RETURN
2050 IF Y+26<A1 OR Y+26>A1+5 THEN 2060 E
LSE Y=A1-26:FA$="N":N=0:JP$="":RETURN
2060 IF Y+26<A4 OR Y+26>A4+5 THEN RETURN
 ELSE Y=A4-26:FA$="N":N=0:JP$="":RETURN
2070 IF Y+26<A2 OR Y+26>A2+9 THEN 2080 E
LSE Y=A2-26:FA$="N":N=0:JP$="":RETURN
2080 IF Y+26<A5 OR Y+26>A5+9 THEN RETURN
 ELSE Y=A5-26:FA$="N":N=0:JP$="":RETURN
2090 IF Y+26<A3 OR Y+26>A3+7 THEN 2100 E
LSE Y=A3-26:FA$="N":N=0:RETURN
2100 IF Y+26<A6 OR Y+26>A6+7 THEN RETURN
 ELSE Y=A6-26:FA$="N":N=0:JP$="":RETURN
2110 GOTO 2110
2120 S$=SPACE$(32)
2130 RESTORE 2520
2140 FOR T=1 TO 10
2150 FOR N=1 TO 16
2160 READ A$
2170 MID$(S$,N,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
2180 MID$(S$,N+16)=CHR$(VAL("&b"+RIGHT$(
A$,8)))
2190 NEXT N
2200 SP$(T)=S$
2210 S$=SPACE$(32)
2220 NEXT T
2230 RESTORE 2520
2240 GOSUB 5410
2250 S$=SPACE$(32)
2260 FOR T=11 TO 20
2270 FOR N=1 TO 16
2280 READ A$
2290 B$=SPACE$(16)
2300 FOR U=1 TO LEN(A$)
```

```
2310 MID$(B$,LEN(A$)+1-U,1)=MID$(A$,U,1)
2320 NEXT U:A$=B$
2330 MID$(S$,N,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
2340 MID$(S$,N+16)=CHR$(VAL("&b"+RIGHT$(
A$,8)))
2350 NEXT N
2360 SP$(T)=S$
2370 S$=SPACE$(32)
2380 NEXT T
2390 RESTORE 4220
2400 S$=SPACE$(32)
2410 FOR T=21 TO 27
2420 FOR N=1 TO 16
2430 READ A$
2440 MID$(S$,N,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
2450 MID$(S$,N+16)=CHR$(VAL("&b"+RIGHT$(
A$,8)))
2460 NEXT N
2470 SP$(T)=S$
2480 S$=SPACE$(32)
2490 NEXT T
2500 RETURN
2510 'Sprite 1
2520 DATA 0000000000000000
2530 DATA 0000000000000000
2540 DATA 0000011110000000
2550 DATA 0000111001000000
2560 DATA 0000111010000000
2570 DATA 0000110000000000
2580 DATA 0000100110000000
2590 DATA 0000000000000000
2600 DATA 0000000000000000
2610 DATA 0000000000000000
2620 DATA 0000010010000010
2630 DATA 0000010001111111
```

```
2640 DATA 0000001000010100
2650 DATA 0000000111001100
2660 DATA 0000000000000000
2670 DATA 0000111111000000
2680 ' Sprite 2
2690 DATA 0000000000000000
2700 DATA 0000000000000000
2710 DATA 0000000000000000
2720 DATA 0000000000000000
2730 DATA 0000000000000000
2740 DATA 0000000000000000
2750 DATA 0000000000000000
2760 DATA 0000000000000000
2770 DATA 0000111100000000
2780 DATA 0000111110000000
2790 DATA 0000101101000000
2800 DATA 0000101111000000
2810 DATA 0000110111000000
2820 DATA 0000111000000000
2830 DATA 0000111111000000
2840 DATA 0000000000000000
2850 ' Sprite 3
2860 DATA 0000000000000000
2870 DATA 0000000000000000
2880 DATA 0000000000000000
2890 DATA 0000000110000000
2900 DATA 0000001100000000
2910 DATA 0000001111000000
2920 DATA 0000011000000000
2930 DATA 0000011100000000
2940 DATA 0000000000000000
2950 DATA 0011000000000000
2960 DATA 0111000000000000
2970 DATA 0111000000000000
2980 DATA 0111000000101000
2990 DATA 0111000000110000
3000 DATA 0111000000000000
```

```
3010 DATA 0000000000000000
3020 ' Sprite 4
3030 DATA 0000111000000000
3040 DATA 0000011100000000
3050 DATA 0000101010000000
3060 DATA 0000111110000000
3070 DATA 0000011100100000
3080 DATA 0011111111100000
3090 DATA 0010111110000000
3100 DATA 0000111100000000
3110 DATA 0000011100000000
3120 DATA 0000001110000000
3130 DATA 0000000000000000
3140 DATA 0000000000000000
3150 DATA 0000000000000000
3160 DATA 0000000000000000
3170 DATA 0000000000000000
3180 DATA 0000000000000000
3190 ' Sprite 5
3200 DATA 0000111111100000
3210 DATA 0000111111100000
3220 DATA 0000111111110000
3230 DATA 0000011111111000
3240 DATA 0000011101111100
3250 DATA 0000001100011100
3260 DATA 0000011100111000
3270 DATA 0000011001110000
3280 DATA 0000011000100000
3290 DATA 0000011100010000
3300 DATA 0000000000000000
3310 DATA 0000000000000000
3320 DATA 0000000000000000
3330 DATA 0000000000000000
3340 DATA 0000000000000000
3350 DATA 0000000000000000
3360 ' Sprite 6
3370 DATA 0000111111000000
```

```
3380 DATA 0000111111100000
3390 DATA 0000111111100000
3400 DATA 0000011111100000
3410 DATA 0000011111100000
3420 DATA 0111111101100000
3430 DATA 0111110001100000
3440 DATA 0100000011000000
3450 DATA 0000000011000000
3460 DATA 0000000011100000
3470 DATA 0000000000000000
3480 DATA 0000000000000000
3490 DATA 0000000000000000
3500 DATA 0000000000000000
3510 DATA 0000000000000000
3520 DATA 0000000000000000
3530 ' Sprite 7
3540 DATA 0000111111000000
3550 DATA 0000111111000000
3560 DATA 0000111111000000
3570 DATA 0000111111000000
3580 DATA 0000011110000000
3590 DATA 0000011110000000
3600 DATA 0000001110000000
3610 DATA 0000001100000000
3620 DATA 0000001100000000
3630 DATA 0000001110000000
3640 DATA 0000000000000000
3650 DATA 0000000000000000
3660 DATA 0000000000000000
3670 DATA 0000000000000000
3680 DATA 0000000000000000
3690 DATA 0000000000000000
3700 ' Sprite 8
3710 DATA 0000111111000000
3720 DATA 0001111111100000
3730 DATA 0001111111110000
3740 DATA 0000111011110000
```

```
3750 DATA 0000000111100000
3760 DATA 0000011111000000
3770 DATA 0000011000000000
3780 DATA 0000010000000000
3790 DATA 0000000000000000
3800 DATA 0000000000000000
3810 DATA 0000000000000000
3820 DATA 0000000000000000
3830 DATA 0000000000000000
3840 DATA 0000000000000000
3850 DATA 0000000000000000
3860 DATA 0000000000000000
3870 ' Sprite 9
3880 DATA 0000111111110000
3890 DATA 0000111111110000
3900 DATA 0000011111100000
3910 DATA 0000011111100000
3920 DATA 0000011101100000
3930 DATA 0000011000000000
3940 DATA 0000110000000000
3950 DATA 0000011000000000
3960 DATA 0000000000000000
3970 DATA 0000000000000000
3980 DATA 0000000000000000
3990 DATA 0000000000000000
4000 DATA 0000000000000000
4010 DATA 0000000000000000
4020 DATA 0000000000000000
4030 DATA 0000000000000000
4040 ' Sprite 25
4050 DATA 1111111100000010
4060 DATA 0001111111111001
4070 DATA 0000000000000100
4080 DATA 0000000000010110
4090 DATA 0000000000000110
4100 DATA 0000000000000000
4110 DATA 0000000000000001
```

```
4120 DATA 0000000000000000
4130 DATA 0000000000000000
4140 DATA 0000000000000000
4150 DATA 0000000000000000
4160 DATA 0000000000000000
4170 DATA 0000000000000000
4180 DATA 0000000000000000
4190 DATA 0000000000000000
4200 DATA 0000000000000000
4210 ' Sprite 10 (negro)
4220 DATA 0111111111011110
4230 DATA 0100111111110010
4240 DATA 0000001111000000
4250 DATA 0000011111100000
4260 DATA 0000001111000000
4270 DATA 0000001111000000
4280 DATA 0000001111000000
4290 DATA 0000000000000000
4300 DATA 0000000000000000
4310 DATA 0000000000000000
4320 DATA 0000000000000000
4330 DATA 0000000000000000
4340 DATA 0000000000000000
4350 DATA 0000000000000000
4360 DATA 0000000000000000
4370 DATA 0000000000000000
4380 ' Sprite 11 (rojo)
4390 DATA 0000000000000000
4400 DATA 0011000000001100
4410 DATA 0011000000001100
4420 DATA 0000000000000000
4430 DATA 0000010000100000
4440 DATA 0000010000100000
4450 DATA 0000000000000000
4460 DATA 0000000000000000
4470 DATA 0000001111000000
4480 DATA 0000001111000000
```

```
4490 DATA 0000011111100000
4500 DATA 0000011111100000
4510 DATA 0000010110100000
4520 DATA 0000010110100000
4530 DATA 0000011111100000
4540 DATA 0000000000000000
4550 ' Sprite 12 (azul claro)
4560 DATA 0000000000000000
4570 DATA 0000000000000000
4580 DATA 0000000000000000
4590 DATA 0011000000001100
4600 DATA 0011000000001100
4610 DATA 0011000000001100
4620 DATA 0011001111001100
4630 DATA 0011101111011100
4640 DATA 0001110000111000
4650 DATA 0001110000111000
4660 DATA 0000100000010000
4670 DATA 0000100000010000
4680 DATA 0000101001010000
4690 DATA 0000001001000000
4700 DATA 0000000000000000
4710 DATA 0000000000000000
4720 ' Sprite 22
4730 DATA 1111111111111111
4740 DATA 1001000110001001
4750 DATA 1010101111010101
4760 DATA 1100010110100011
4770 DATA 0111111111111110
4780 DATA 0000000110000000
4790 DATA 0000000000000000
4800 DATA 0000000000000000
4810 DATA 0000000000000000
4820 DATA 0000000000000000
4830 DATA 0000000000000000
4840 DATA 0000000000000000
4850 DATA 0000000000000000
```

```
4860 DATA 0000000000000000
4870 DATA 0000000000000000
4880 DATA 0000000000000000
4890 ' Sprite 23
4900 DATA 0000000000000000
4910 DATA 0000000000000000
4920 DATA 0000000000000000
4930 DATA 0000000000000000
4940 DATA 0000000000000000
4950 DATA 0000000001111111
4960 DATA 0000110011111110
4970 DATA 0001110111111100
4980 DATA 0010111111111000
4990 DATA 0111111111111100
5000 DATA 0001110111111011
5010 DATA 0000000000000000
5020 DATA 0000000000000000
5030 DATA 0000000000000000
5040 DATA 0000000000000000
5050 DATA 0000000000000000
5060 ' Sprite 24
5070 DATA 0000000000000000
5080 DATA 0000000000000000
5090 DATA 0000000000000000
5100 DATA 0000000000000000
5110 DATA 0000000000000000
5120 DATA 0000000000000000
5130 DATA 0000110000000000
5140 DATA 0001110111111000
5150 DATA 0010111111111100
5160 DATA 0111111111111110
5170 DATA 0011110111111111
5180 DATA 0000000000011111
5190 DATA 0000000000000000
5200 DATA 0000000000000000
5210 DATA 0000000000000000
5220 DATA 0000000000000000
```

```
5230 ' Sprite 25
5240 DATA 0001000000000000
5250 DATA 0010100000000000
5260 DATA 0111010000000000
5270 DATA 1111101000000000
5280 DATA 1111011000000000
5290 DATA 0111110000000000
5300 DATA 0011100000000000
5310 DATA 0000000000000000
5320 DATA 0000000000000000
5330 DATA 0000000000000000
5340 DATA 0000000000000000
5350 DATA 0000000000000000
5360 DATA 0000000000000000
5370 DATA 0000000000000000
5380 DATA 0000000000000000
5390 DATA 0000000000000000
5400 REM Grafico del fondo
5410 IF JJK$="J" THEN 5430 ELSE COLOR 1,
3,2:CLS
5420 REM Posicionar cables ascensor
5430 FOR T=10 TO 182 STEP 2
5440 PSET(48,T),1
5450 NEXT T
5460 FOR T=9 TO 181 STEP 2
5470 PSET(47,T),1
5480 NEXT T
5490 FOR T=10 TO 182 STEP 2
5500 PSET(74,T),1
5510 NEXT T
5520 FOR T=9 TO 181 STEP 2
5530 PSET(73,T),1
5540 NEXT T
5550 FOR T=10 TO 182 STEP 2
5560 PSET(100,T),1
5570 NEXT T
5580 FOR T=9 TO 181 STEP 2
```

```
5590 PSET(99,T),1
5600 NEXT T
5610 IF JJK$="J" THEN JJK$="":RETURN
5620 LINE(74-9,184)-(74+9,192),1,BF
5630 LINE(74-9,0)-(74+9,8),1,BF
5640 LINE(48-9,184)-(48+9,192),1,BF
5650 LINE(48-9,0)-(48+9,8),1,BF
5660 LINE(100-9,184)-(100+9,192),1,BF
5670 LINE(100-9,0)-(100+9,8),1,BF
5680 FOR T=1 TO 192 STEP 5
5690 LINE(195,T)-(205,T+5),11,B
5700 NEXT T
5710 DRAW"bm0,60;c12r30"
5720 DRAW"bm5,160;c12r25"
5730 DRAW"bm118,160;c12r15f5r2e5r20f5e5r
12"
5740 DRAW"bm118,60;c12r20f5r2e5r37"
5750 DRAW"bm135,110;c12r25f5e5r17"
5760 DRAW"bm215,60;c12r50"
5770 DRAW"bm215,110;c12r25"
5780 DRAW"bm215,160;c12r25"
5790 DRAW"bm0,20;c2f3e2r1f3r2f2e3f2r2e3r
2f2e2f1e4h3u2h2e3u2erh2e4"
5800 PAINT(10,10),2
5810 DRAW"bm0,61;c2r30f2d2g3d212g2f3r3f3
g3f2g4d1f2r2f2d2g3d1g312h2g3f3d2f2r2f2d2
g2f3g5d211h311g312h2g413h2g311d2f3d2f1d2
g3d1g2d1f3d1f3d2g3d4r26f2g3d211g2f4d2g2f
3d2g2f3d1f1d1"
5820 PAINT(10,100),2
5830 DRAW"bm115,0;c2f3d2f2g3d2g2d2f1g2f4
r4e2r1f3r1e3u2e3r2f3g1f2r2e3r1f2e3h2e3r2
f2r3f2e3r2f3g2f2e3u2e2r3d2g2f4r2e3r2u2e1
h3u2e3u1e2h2e3"
5840 PAINT(150,5),2
5850 DRAW"bm215,0;c2f3d2g2d1g1d2f2g2f4r2
f2r2e3h2e3f2r1f2e3r2f3r2f2r2e2f3e2f1"
```

```
5860 PAINT(250,5),2
5870 DRAW"bm212,192;e3u2e2u1h2u2e3h2u2h3
e2h3u2e2r27u2e2u3e2u2e2u3h2u1e2u2e3u2e2u
1h3l2h3l2g2h3l1h2g3l2g2l2h3g3hu1h2u2e2u1
h4u1e1r25u2e2u3e3u2e3u3h3u2h2u2e2h3l2h3g
2h3e2h3g3l2g2l2h2g3h3u2e3u1h3e2h1r50"
5880 PAINT(250,185),2
5890 DRAW"bm115,192;c2e3u2e3u1h3u2e2h3l2
h1u2e3h2u1e3r15f5r2e5r20f5e5r12f2d2g2d2g
3l1h2g3d2f3d2f3r2d2f3d2f1"
5900 PAINT(160,170),2
5910 DRAW"bm187,61;c2l37g5l2h5l20df2r2f2
d3g3l2g3d2f3g2d2f3g2d2g2f3d2g4f3e2f3d3g3
d2g3d3f3r2e3r1f2r2f3e2h2e1r3f3r2f4e3u1e2
r1f2g1f2r1e3r2e2r2f3r1e2r2f2r1f2r2e3u1e3
u2e2u2e1h1l18g5h5l25u2h2u1h3u2e1u2e2u1h2
u2h1e2u2e3u2e3u1h2e2r2f3r2f2r2e2r1f2r2e3
r1f2r2f2r1e2r1
5920 DRAW"bm187,61;c2f2d4g3d2f2d1g3l2h3e
2h3l2g3f2g2h2l1g2"
5930 PAINT(126,63),2
5940 DRAW"bm227,159;c1r8l2u1l4r1u8l3d1u2
r3u2r1u2l3d1u1r1u1r4f2d7r1d1r2e1g1l2u1l1
u2l1u6l3d13r2l1u11"
5950 LINE(0,182)-(256,192),1,BF
5960 DRAW "bm 3,185
5970 ZEIT=5000
5980 COLOR 15
5990 PRINT #1,"MAN:   TIME:";
6000 PRINT #1,USING "####";ZEIT;
6010 PRINT #1," WATER"
6020 COLOR 1
6030 DI$="c15h4u1e2r4f2d1g3h1u1l1h1l1r1u
1e1f1e1f1d1r1l1g1l1"
6040 VFL=6
6050 AR$="GH"
6060 COLOR 15
```

```
6070 GOSUB 6270
6080 COLOR 1
6090 RETURN
6100 REM Linea de vuelo del fantasma
6110 DATA "U",9,"L",6,"R",6,"U",5,"L",19
,"R",3,"O",10,"R",16,"U",14,"E",0
6120 RESTORE 6110
6130 FOR T=1 TO 10
6140 READ H$(T),H(T)
6150 NEXT T
6160 REM Posiciones de diamantes
6170 DATA 15,159,125,159,146,109,230,109
,130,59
6180 RESTORE 6170
6190 FOR T=1 TO 5
6200 READ PX(T),PY(T)
6210 NEXT T
6220 FOR T=1 TO 5
6230 P$(T)="N"
6240 NEXT T
6250 RETURN
6260 REM Cuando se ha perdido
         un hombrecito
6270 VFL=VFL-1
6280 IF VFL=0 THEN LINE(35,185)-(46,192)
,1,BF:DRAW "bm 30,185":PRINT #1,VFL:LINE
 (90,94)-(90+16*6,106),3,BF:DRAW "bm 90,
96":PRINT #1,"  GAME OVER  ":INTERVAL OF
F:GOTO 7010
6290 LINE(35,185)-(46,192),1,BF
6300 DRAW "bm 30,185"
6310 PRINT #1,VFL
6320 IF AR$="GH" THEN AR$="":RETURN
6330 GOTO 400
6340 RETURN
6350 REM Recargar
6360 IF X>220 AND X<230 AND Y>130 AND Y<
140 THEN GOSUB 6700:RETURN ELSE SCHUSS=1
:RETURN
```

```
6370 REM Disparar
6380 IF LE$="J" THEN RETURN
6390 IF WA$="LEER" THEN RETURN
6400 IF R$="r" THEN PUT SPRITE 20,(X+16,
Y+10),4,10:GOSUB 6450 ELSE PUT SPRITE 20
,(X-16,Y+10),4,20:GOSUB 6470
6410 SCHUSS=0
6420 GOSUB 6660
6430 PUT SPRITE 20,(0,0),0,31
6440 RETURN
6450 IF O>X+14 AND O<X+33 AND P>Y+8 AND
P<Y+19 THEN L=1:L1=0:O=194:P=-1:ZEIT=ZEI
T+100:RETURN
6460 RETURN
6470 IF O>X-17 AND O<X+2 AND P>Y+8 AND P
<Y+19 THEN L=1:L1=0:O=194:P=-1:ZEIT=ZEIT
+100:RETURN
6480 RETURN
6490 SCHUSS=0
6500 RETURN
6510 REM Parar ascensores
6520 LI$="N"
6530 TR$=STR$(A1)
6540 DRAW "bm40,"+TR$+";c1"+Q$
6550 PUT SPRITE 10,(0,0),15,31
6560 TR$=STR$(A3)
6570 DRAW "bm92,"+TR$+";c1"+Q$
6580 PUT SPRITE 12,(0,0),15,31
6590 TR$=STR$(A4)
6600 DRAW "bm40,"+TR$+";c1"+Q$
6610 PUT SPRITE 13,(0,0),15,31
6620 TR$=STR$(A6)
6630 DRAW "bm92,"+TR$+";c1"+Q$
```

```
6640 PUT SPRITE 15,(255,191),15,31
6650 RETURN
6660 IF WA=51 THEN WA$="LEER":RETURN
6670 LINE(248-WA,186)-(248,188),1,BF
6680 WA=WA+1
6690 RETURN
6700 IF WA$<>"LEER" THEN RETURN
6710 WA$=""
6720 FOR T=1 TO 50
6730 PSET(248-T,186),2
6740 PSET(248-T,187),7
6750 PSET(248-T,188),2
6760 NEXT T
6770 WA=0
6780 RETURN
6790 ZEIT=ZEIT-100
6800 IF ZEIT<=0 THEN 6270 ELSE LINE(96,1
85)-(128,192),1,BF:DRAW "bm 100,185":COL
OR 15:PRINT #1,USING "####";ZEIT:RETURN
6810 REM Grafico de entrada
6820 SCREEN 2
6830 COLOR 15,1,1
6840 CLS
6850 DRAW "bm60,40;c15d30u30f10e10d30"
6860 DRAW "bm90,70;c15u30r15d15l15"
6870 LINE(90,55)-(105,70),15
6880 DRAW"bm80,90;d30u30f10e10d30"
6890 DRAW"bm110,90;d30"
6900 DRAW"bm120,90;d30"
6910 DRAW"bm140,90;d30"
6920 LINE(120,90)-(140,120),15
6930 DRAW"bm150,90;r20l20d15r10l10d15r20
"
6940 DRAW"bm180,120;u30r15d15l15"
6950 LINE(180,105)-(195,120),15
6960 DRAW "bm 5,160"
```

```
6970 PRINT #1,"<c> by ARNE y DANIEL STOF
FREGEN"
6980 FOR T=1 TO 1000
6990 NEXT T
7000 RETURN
7010 FOR T=1 TO 2000
7020 NEXT T
7030 COLOR 15,1,1
7040 SCREEN 0
7050 LOCATE 10,12,0
7060 PRINT "ANOTHER GAME  ?"
7070 LOCATE 10,15
7080 PRINT "<press ENTER>"
7090 INPUT A$
7100 RUN
```

Descripción de rutinas 1 a 3 en lenguaje máquina

Además del considerable léxico BASIC, el ordenador MSX ofrece también la posibilidad de programar en lenguaje máquina, para lo cual necesitamos un ensamblador o ASSEMBLER.

En este libro no hemos incluido ningún ASSEMBLER, por lo que debemos teclear los datos en lenguaje máquina en forma de cifras hexadecimales. De todas formas, en los tres ejemplos siguientes explicamos detalladamente el significado de todas las cifras a entrar.

¿De qué tratan los listados?

La rutina 1 en lenguaje máquina le transmite una primera impresión de la rapidez del lenguaje máquina: toda la pantalla se llena de la letra 'M' en menos de un segundo. También puede modificar esta rutina de modo que el ordenador escriba un carácter determinado en toda la pantalla gráfica lo más rápidamente posible o ... las aclaraciones del propio listado le permitirán experimentar todas las aplicaciones de esta rutina.

La rutina 2 en lenguaje máquina le muestra la forma de copiar en la RAM una zona de la VRAM en fracciones de segundos. Nuestro ejemplo incluso llega a copiar la VRAM entera en la RAM. Después de haber efectuado el proceso puede volver tranquilamente a la pantalla de texto, porque ...

con la rutina 3 en lenguaje máquina puede volver a reclamar la pantalla recién almacenada en la RAM a la VRAM. De esta forma, tal como muestra nuestro ejemplo, podría generar un gráfico, almacenarlo en la RAM mediante la rutina 2 en lenguaje máquina, crear un segundo dibujo y entretanto reclamar de nuevo el gráfico memorizado anteriormente. ¡Recuerde que la memoria RAM es limitada! Si copia sólo partes de su gráfico, las posibilidades de aplicaciones también son ilimitadas.

```
10 REM Demostrar utilizacion de rutinas
       en lenguaje maquina en la MSX-ROM
20 REM 1. &H0056
          Llenar una seccion de la
          memoria VRAM con un
          caracter
30 CLEAR 200,39999!
40 SCREEN 0
50 COLOR 1,15
60 WIDTH 40
70 FOR N=40000! TO 40011!
80 READ A$
90 POKE N,VAL("&H"+A$)
100 NEXT N
110 REM &H4D es el numero del caracter
       (decimal=77), que corresponde
       a la letra 'M'
120 DATA 3E,4D
130 REM &H00 y &H00 indican el inicio
       del llenado de la VRAM
       (decimal=0)
140 DATA 21,00,00
150 REM &HC0 y &H03 indican la longitud
       de la seccion de memoria, que
       debe llenarse con caracteres
       (decimal=960)
160 DATA 01,C0,03
170 REM &H56 y &H00 es la direccion
       inicial de la rutina en lenguaje
       maquina de la VROM, que debe
       llenar una seccion de la memoria
       antes indicada (decimal=86)
180 DATA CD,56,00
190 REM Zur}ck ins BASIC
200 DATA C9
210 DEFUSR=40000!
220 A=USR(0)
```

```
10 REM Demonstracion del uso de rutinas
      en lenguaje maquina en la MSX-ROM
20 REM 2. &H0059
         Copiar una seccion de
         memoria de la VRAM a
         la RAM
30 CLEAR 200,39999!
40 SCREEN 0
50 COLOR 1,15
60 WIDTH 40
70 FOR N=40000! TO 40012!
80 READ A$
90 POKE N,VAL("&H"+A$)
100 NEXT N
110 REM &H00 y &H00 indican la primera
        direccion de la VRAM, que debe
        ser transferida a la RAM
        (decimal=0)
120 DATA 21,00,00 'HL
130 REM &H00 y &H40 indican la longitud
        de las posiciones de memoria,
        que deben ser transferidas
        de la VRAM a la RAM
        (decimal=16384)
140 DATA 01,00,40 'BC
150 REM &H28 y &HA0 indican la primera
        direccion de la RAM, hacia la
        cual deben fluir los va-
        lores de la VRAM
        (decimal=41000)
160 DATA 11,28,A0 'DE
170 REM &H59 y &H00 indican la rutina
        en lenguaje maquina de la ROM,
        que nos ayuda a transferir los
        datos (decimal=89)
180 DATA CD,59,00 'Call
190 REM Vuelta al BASIC
200 DATA C9
210 DEFUSR=40000!
220 A=USR(0)
```

```
220 REM 3. Creacion de un dibujo y su
        almacenamiento en la RAM.
230 REM Desarrollo de una rutina de re-
        cuperacion, que vuelve a
        transportar los datos de
        la RAM a la VRAM
240 SCREEN 2
250 FOR N=10 TO 80 STEP 5
260 CIRCLE(128,96),N
270 NEXT N
280 FOR N=132 TO 255 STEP 10
290 PAINT(N,96)
300 NEXT N
310 A=USR(0)
320 CLS
330 FOR N=40000! TO 40012!
340 READ A$
350 POKE N,VAL("&H"+A$)
360 NEXT N
370 REM &H28 y &HA0 indican la primera
        direccion de la RAM, que debe
        ser transferida a la VRAM
        (decimal=41000)
380 DATA 21,28,A0 'HL
390 REM &H00 y &H40 indican la longitud
        de las posiciones de memoria,
        que deben ser transferidas
        de la RAM a la VRAM
        (decimal=16384)
400 DATA 01,00,40 'BC
410 REM &H00 y &H00 indican la primera
        direccion de la VRAM, hacia
        la cual deben fluir los
        valores de la RAM
        (decimal=0)
420 DATA 11,00,00 'DE
```

```
430 REM &H5C y &H00 indican la rutina en
        lenguaje maquina de la ROM, que
        debe ayudarnos a transferir
        los datos (decimal=92)
440 DATA CD,5C,00 'Call
450 REM Vuelta al BASIC
460 DATA C9
470 DEFUSR=40000!
480 CLS
490 FOR N=255 TO 10 STEP -10
500 LINE(0,0)-(N,N),INT(RND(1)*14+1),BF
510 NEXT N
520 A=USR(0)
530 GOTO 530
```

Direcciones importantes de RAM y ROM de los ordenadores MSX

Advertencia:

Cálculo de un byte doble, por ejemplo 'A y B':

B * 256 + A

&H1BBF a &H1BBF+2047

Juego de caracteres que es copiado en la VRAM al conectar el ordenador (como muestra el apéndice). Esta dirección se encuentra además en las posiciones de memoria &HF920 y &HF921.

&HF3AE

En esta posición se almacena la anchura actual de la pantalla (caracteres por línea) de SCREEN 0 (al conectar = 37). Si usted cambia este valor, no podrá reconocer el proceso en la pantalla mientras no entre el comando SCREEN 0.

&HF3AF

Esta posición de memoria almacena la anchura actual de la pantalla (caracteres por línea) de SCREEN 1 (al conectar = 29). Si modifica este valor, podrá observar el proceso a través del comando SCREEN 1 en la pantalla.

&HF3B0

Esta posición de memoria almacena la anchura actual de la pantalla (caracteres por línea) del SCREEN en que está trabajando en ese momento. Si cambia este valor, el nuevo estado indicado (p.e. '30'=30 caracteres por línea) ya toma forma en la línea siguiente a la de posición actual del cursor. En este caso no se limpia la pantalla, a diferencia del comando WIDTH.

$HF3B1

Esta posición memoriza la cantidad de líneas por página (24 en caso normal). Puede cambiarse este valor, que no será borrado al entrar un comando SCREEN. Por lo tanto, dicha entrada afecta simultáneamente a ambas pantallas de texto.

&HF3B3 y &HF3B4

Corresponde a BASE(0) (tabla de nombres de SCREEN 0)

&HF3B7 y &HF3B8

Corresponde a BASE(2) (tabla del generador de patrones de SCREEN 0)

&HF3BD y &HF3BE

Corresponde a BASE(5) (tabla de nombres de SCREEN 1)

&HF3BF y &HF3C0

Corresponde a BASE(6) (tabla de colores de SCREEN1)

&HF3C1 y &HF3C2

Corresponde a BASE(7) (tabla del generador de patrones de SCREEN 1)

&HF3C3 y &HF3C4

Corresponde a BASE(8) (tabla de atributos de sprites de SCREEN 1)

&HF3C5 y &HF3C6

Corresponde a BASE(9) (tabla de patrones de sprites de SCREEN 1)

&HF3C7 y &HF3C8

Corresponde a BASE(10) (tabla de nombres de SCREEN 2)

&HF3C9 y &HF3CA

Corresponde a BASE(11) (tabla de colores de SCREEN 2)

&HF3CB y &HF3CC

Corresponde a BASE(12) (tabla del generador de patrones de SCREEN 2)

&HF3CD y &HF3CE

Corresponde a BASE(13) (tabla de atributos de sprites de SCREEN 2)

&HF3CF y &HF3D0

Corresponde a BASE(14) (tabla de patrones de sprites de SCREEN 2)

&HF3D1 y &HF3D2

Corresponde a BASE(15) (tabla de nombres de SCREEN 3)

&HF3D5 y &HF3D6

Corresponde a BASE(17) (tabla del generador de patrones de SCREEN 3)

&HF3D7 y &HF3D8

Corresponde a BASE(18) (tabla de atributos de sprites de SCREEN 3)

&HF3D9 y &HF3DA

Corresponde a BASE(19) (tabla de patrones de sprites de SCREEN 3)

&HF3DB

Esta posición memoriza el valor 0 cuando se ha desactivado previamente el chasquido del teclado mediante el comando SCREEN (tercer parámetro), o el valor 1 después de activar este sonido. Además de leer su estado, también puede activar y desactivar el chasquido POKEando directamente en esta posición de memoria.

&HF3DC

Esta posición memoriza la actual posición Y del cursor (la línea en que se encuentra), y puede utilizarse para posicionar el cursor, parecido al comando LOCATE.

&HF3DD

Esta posición memoriza la actual posición X del cursor (la columna en que se encuentra), y puede utilizarse para posicionar el cursor, parecido al comando LOCATE.

&HF3DE

En esta posición de memoria se almacena un número que nos informa si se indican las teclas de funciones (corresponde a KEY ON) o no (corresponde a KEY OFF). El valor almacenado es 255 cuando se visualizan las teclas de funciones, y 0 en el caso contrario. Si POKEa el correspondiente valor en esta posición de memoria, el cambio no se notará hasta limpiar a continuación la pantalla con CLS.

&HF3DF

Corresponde al contenido actual del registro VDP(0)

&HF3E0

Corresponde al contenido actual del registro VDP(1)

&HF3E1

Corresponde al contenido actual del registro VDP(2)

&HF3E2

Corresponde al contenido actual del registro VDP(3)

&HF3E3

Corresponde al contenido actual del registro VDP(4)

&HF3E4

Corresponde al contenido actual del registro VDP(5)

&HF3E5

Corresponde al contenido actual del registro VDP(6)

&HF3E6

Corresponde al contenido actual del registro VDP(7)

&HF3E7

Corresponde al contenido actual del registro VDP(8)

&HF3E9

Esta posición de memoria almacena un número que informa sobre el actual color del primer plano (corresponde al color de los caracteres). Si usted POKEa otro valor numérico en esta posición, no tendrá validez, es decir el ordenador no lee e interpreta su contenido, hasta que entre un comando SCREEN.

&HF3EA

Esta posición de memoria almacena un número que informa sobre el actual color del fondo. Si usted POKEa otro valor en esta posición, no tendrá validez, es decir el ordenador no lee e interpreta el nuevo contenido, hasta que entre un comando SCREEN.

&HF3EB

Esta posición memoriza un número que da información sobre el actual color del área de los bordes (sólo válido en SCREEN 1, 2 y 3). Si POKEa otro valor en esta posición, no será válida, es decir el ordenador no lee e interpreta el nuevo contenido, hasta que entre un comando SCREEN.

&HF415

La lectura de esta posición de memoria equivale al comando LPOS. En ella el ordenador MSX memoriza la posición en la cual se encuentra actualmente la cabeza de la impresora conectada.

&HF417

La lectura de esta posición de memoria equivale al valor indicado en el quinto parámetro del comando SCREEN (impresora MSX sí/no). Si el valor es 0, la impresora conectada debe ser MSX; si el valor es distinto de 0, los caracteres gráficos específicos del MSX serán emitidos a la impresora en forma de espacios.

&HF87F a &HF87F+159

Estas 160 posiciones de memoria almacenan el contenido actual de las teclas de función. Por lo tanto, mediante PEEK puede consultar directamente el contenido de estas teclas. También es posible modificar su contenido a través de las posiciones de memoria mediante el comando POKE. Sin embargo, el cambio efectuado no será visible hasta después de pulsar

la tecla SHIFT. Si desea visualizar el nuevo contenido de las teclas de función instantáneamente, deberá POKEarlo a la RAM de vídeo adecuada (con VPOKE). Mediante VPEEK podrá leer además el actual contenido de dichas teclas en las pantallas de texto (mientras la pantalla tenga anchura suficiente para poder representar el máximo número de caracteres de las funciones).

&HF922 y &HF923

Estas posiciones memorizan la dirección actual que hace referencia al principio de la tabla de nombres. De forma parecida al comando VDP, usted tiene la posibilidad de leer dicho contenido (que corresponde a un valor BASE) así como modificarlo instantáneamente mediante el comando POKE. Sin embargo, es aconsejable obtener informaciones precisas antes de tocar estas posiciones de memoria (consulte capítulos BASE, SCREEN y VDP).

&HF924 y &HF925

En estas posiciones de memoria se almacena la actual dirección que hace referencia al principio de la tabla del generador de patrones. De forma parecida al comando VDP, usted tiene la posibilidad de leer dicho contenido (que corresponde a un valor BASE) y de modificarlo además instantáneamente mediante POKE. Sin embargo, es aconsejable obtener informaciones precisas antes de tocar dichas posiciones de memoria (consulte capítulos BASE, SCREEN y VDP).

&HF926 y &HF927

Estas posiciones de memoria contienen la actual dirección que hace referencia al principio de la tabla de patrones de sprites. De forma parecida al comando VDP, usted tiene la posibilidad de leer dicho contenido (que corresponde a un valor BASE) y de modificarlo además instantáneamente mediante POKE. Sin embargo, es aconsejable conseguir informaciones precisas antes de tocar esta posición de

memoria (consulte capítulos BASE, SCREEN y VDP).

&HF928 y &HF929

Estas posiciones de memoria contienen la actual dirección que hace referencia al principio de la tabla de atributos de sprites. De forma parecida al comando VDP, usted tiene la posibilidad de leer dicho contenido (que corresponde a un valor BASE) y de modificarlo además instantáneamente mediante el comando POKE. Sin embargo, es aconsejable obtener informaciones precisas antes de tocar esta posición de memoria (consulte capítulos BASE, SCREEN y VDP).

&HF956 y siguientes

Esta es la zona de memoria requerida por el comando PLAY para memorizar las notas entradas anteriormente. Desde estas posiciones el ordenador lee sonido por sonido a través de una rutina de interrupciones. En el capítulo PLAY(N) encontrará más información sobre algunas de estas posiciones de memoria.

&HFBCC

Usted recordará que el cursor cambia de aspecto al moverlo a un punto de la pantalla ocupado por otro carácter. Esta posición de memoria contiene precisamente el carácter que se encuentra bajo el cursor en ese momento (este carácter se representa en modo inverso).

&HFC9D o HFC9C

Aquí se memorizan las coordenadas X e Y del 'pad' conectado (el comando PAD del BASIC MSX permite leer esta posición de memoria).

&HFCA9

El valor de esta posición corresponde al valor indicado a través del tercer parámetro del comando LOCATE (¿desea que

cursor aparezca en la pantalla durante la ejecución del programa? sí=1 no=0). Por lo tanto, también puede modificar dicho parámetro a través de esta posición de memoria.

&HFCAB

Esta posición contiene un '0' si desea trabajar con letras mayúsculas y minúsculas (las mayúsculas se obtienen pulsando simultáneamente la tecla SHIFT) o un '1' si trabaja con mayúsculas. POKEando un 1 en esta posición de memoria escribirá en letra mayúscula a partir de ese momento, sin necesidad de pulsar previamente la tecla CAPS (sin embargo, de esta forma no se encenderá el LED de la tecla CAPS).

&HFCAF

En ella se memoriza el número del SCREEN actual.

&HFCB0

Aquí se memoriza el número del último SCREEN seleccionado. De esta forma, a través de esta posición de memoria, podemos saber cuál fue la pantalla de texto seleccionada antes de trabajar en la actual pantalla gráfica.

&HFCB2

Utilizando el comando PAINT en SCREEN 3, usted no sólo tiene la posibilidad de indicar el color del relleno, sino también el color hasta el cual debe llegar a colorear. Dichas informaciones se almacenan en esta posición de memoria.

&HFCB3 o &HFCB5 o &HFCB7 o &HFCB9

Las dos primeras posiciones memorizan los datos referentes a las coordenadas X-Y actuales del cursor gráfico. Las dos posiciones siguientes contienen las informaciones necesarias para saber si -y de qué forma- se distingue un cursor gráfico relativo del curosr gráfico absoluto a través de un comando relativo (DRAW con 'N', comando STEP, distancia de

los puntos del círculo al centro).

&HFCBC

El comando DRAW permite seleccionar diferentes escalas con ayuda del parámetro 'S' y de un valor numérico. La actual ampliación/reducción (4 es el estado original y corresponde a la escala entrada:emisión = 1:1) es almacenada en esta posición de memoria, y puede ser modificado mediante POKE.

&HFCBD

El comando DRAW permite seleccionar diversos ángulos en pasos de 90 grados indicando el parámetro 'A' y un valor numérico. El giro actual (0 = no giro, que corresponde al estado original) es memorizado en esta posición, y puede ser modificado mediante POKE.

Posibles valores que podemos entrar en el registro 2 de VDP

==

00	0000	01	0400	02	0800	03	0C00
04	1000	05	1400	06	1800	07	1C00
08	2000	09	2400	0A	2800	0B	2C00
0C	3000	0D	3400	0E	3800	0F	3C00

Posibles valores que podemos entrar en el registro 4 de VDP

==

00	0000	01	0800	02	1000	03	1800
04	2000	05	2800	06	3000	07	3800

Posibles valores que podemos entrar en el registro 5 de VDP

==

00	0000	01	0080	02	0100	03	0180
04	0200	05	0280	06	0300	07	0380
08	0400	09	0480	0A	0500	0B	0580
0C	0600	0D	0680	0E	0700	0F	0780
10	0800	11	0880	12	0900	13	0980
14	0A00	15	0A80	16	0B00	17	0B80
18	0C00	19	0C80	1A	0D00	1B	0D80
1C	0E00	1D	0E80	1E	0F00	1F	0F80
20	1000	21	1080	22	1100	23	1180
24	1200	25	1280	26	1300	27	1380
28	1400	29	1480	2A	1500	2B	1580
2C	1600	2D	1680	2E	1700	2F	1780
30	1800	31	1880	32	1900	33	1980
34	1A00	35	1A80	36	1B00	37	1B80
38	1C00	39	1C80	3A	1D00	3B	1D80
3C	1E00	3D	1E80	3E	1F00	3F	1F80
40	2000	41	2080	42	2100	43	2180
44	2200	45	2280	46	2300	47	2380
48	2400	49	2480	4A	2500	4B	2580
4C	2600	4D	2680	4E	2700	4F	2780
50	2800	51	2880	52	2900	53	2980
54	2A00	55	2A80	56	2B00	57	2B80
58	2C00	59	2C80	5A	2D00	5B	2D80
5C	2E00	5D	2E80	5E	2F00	5F	2F80
60	3000	61	3080	62	3100	63	3180
64	3200	65	3280	66	3300	67	3380
68	3400	69	3480	6A	3500	6B	3580
6C	3600	6D	3680	6E	3700	6F	3780
70	3800	71	3880	72	3900	73	3980
74	3A00	75	3A80	76	3B00	77	3B80
78	3C00	79	3C80	7A	3D00	7B	3D80
7C	3E00	7D	3E80	7E	3F00	7F	3F80

Posibles valores que podemos entrar en el registro 6 de VDP

==

00	0000	01	0800	02	1000	03	1800
04	2000	05	2800	06	3000	07	3800

Posibles valores que podemos entrar en el registro 3 de VDP
==

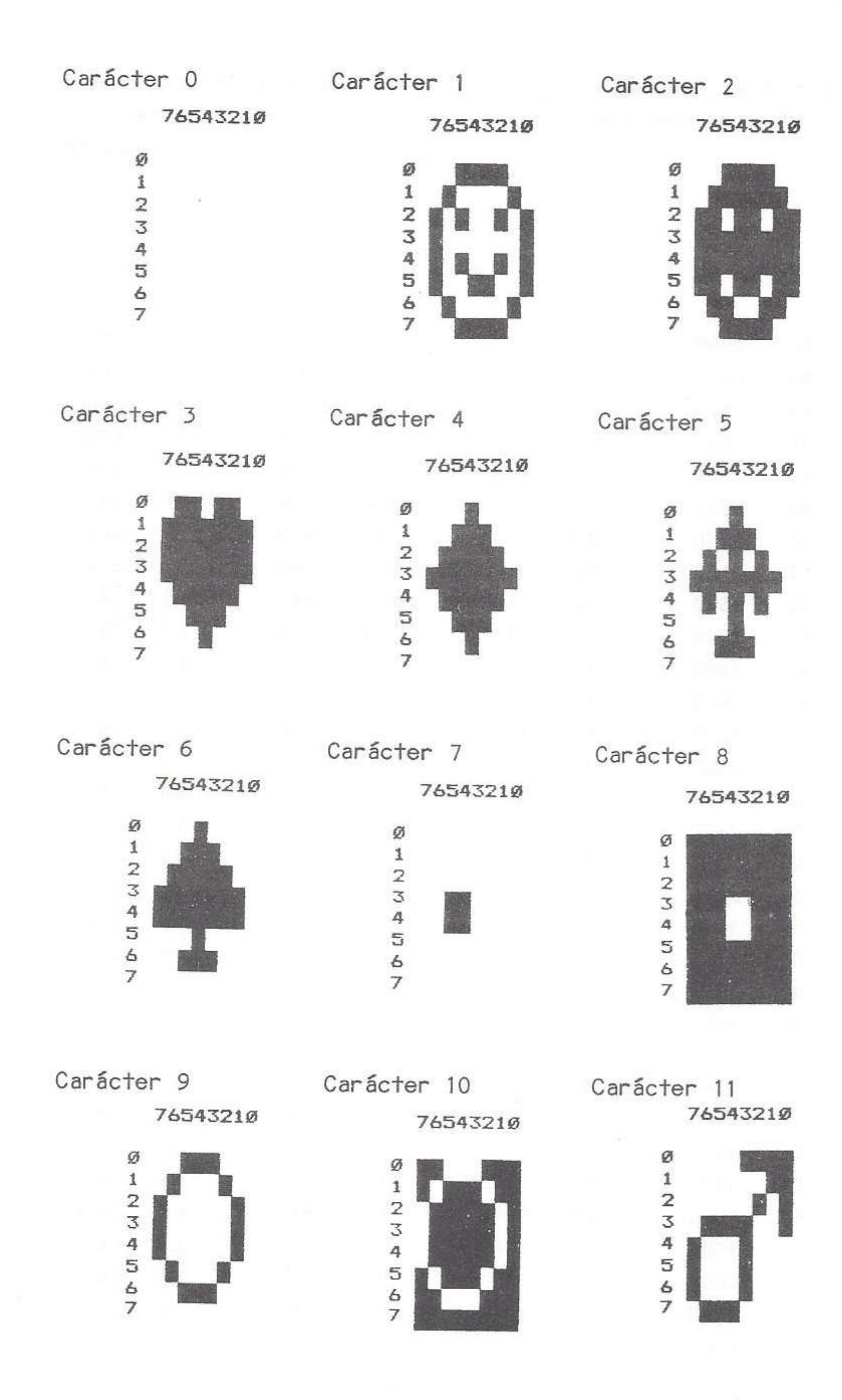
parte 1

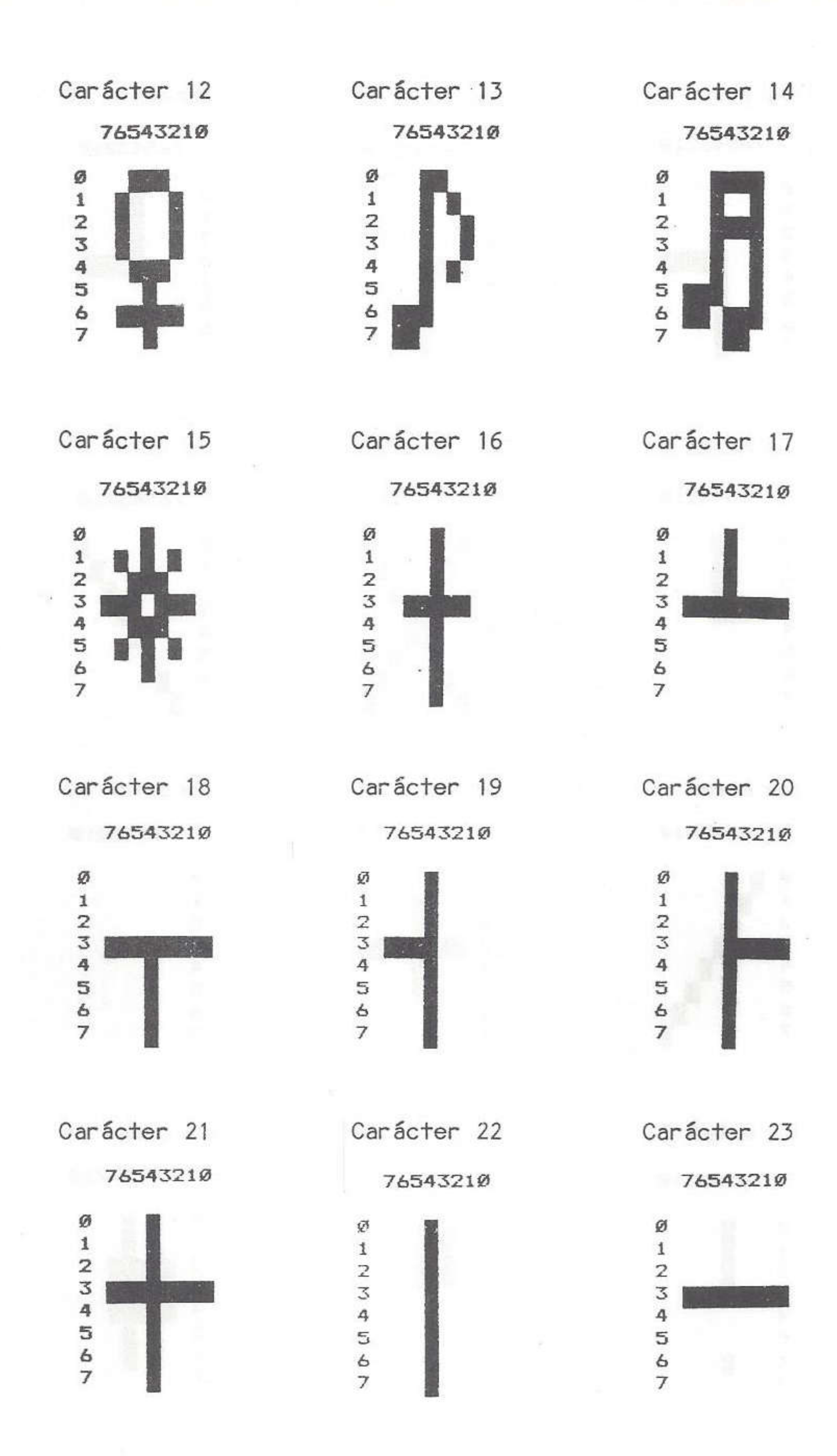
00	0000	01	0040	02	0080	03	00C0
04	0100	05	0140	06	0180	07	01C0
08	0200	09	0240	0A	0280	0B	02C0
0C	0300	0D	0340	0E	0380	0F	03C0
10	0400	11	0440	12	0480	13	04C0
14	0500	15	0540	16	0580	17	05C0
18	0600	19	0640	1A	0680	1B	06C0
1C	0700	1D	0740	1E	0780	1F	07C0
20	0800	21	0840	22	0880	23	08C0
24	0900	25	0940	26	0980	27	09C0
28	0A00	29	0A40	2A	0A80	2B	0AC0
2C	0B00	2D	0B40	2E	0B80	2F	0BC0
30	0C00	31	0C40	32	0C80	33	0CC0
34	0D00	35	0D40	36	0D80	37	0DC0
38	0E00	39	0E40	3A	0E80	3B	0EC0
3C	0F00	3D	0F40	3E	0F80	3F	0FC0
40	1000	41	1040	42	1080	43	10C0
44	1100	45	1140	46	1180	47	11C0
48	1200	49	1240	4A	1280	4B	12C0
4C	1300	4D	1340	4E	1380	4F	13C0
50	1400	51	1440	52	1480	53	14C0
54	1500	55	1540	56	1580	57	15C0
58	1600	59	1640	5A	1680	5B	16C0
5C	1700	5D	1740	5E	1780	5F	17C0
60	1800	61	1840	62	1880	63	18C0
64	1900	65	1940	66	1980	67	19C0
68	1A00	69	1A40	6A	1A80	6B	1AC0
6C	1B00	6D	1B40	6E	1B80	6F	1BC0
70	1C00	71	1C40	72	1C80	73	1CC0
74	1D00	75	1D40	76	1D80	77	1DC0
78	1E00	79	1E40	7A	1E80	7B	1EC0
7C	1F00	7D	1F40	7E	1F80	7F	1FC0

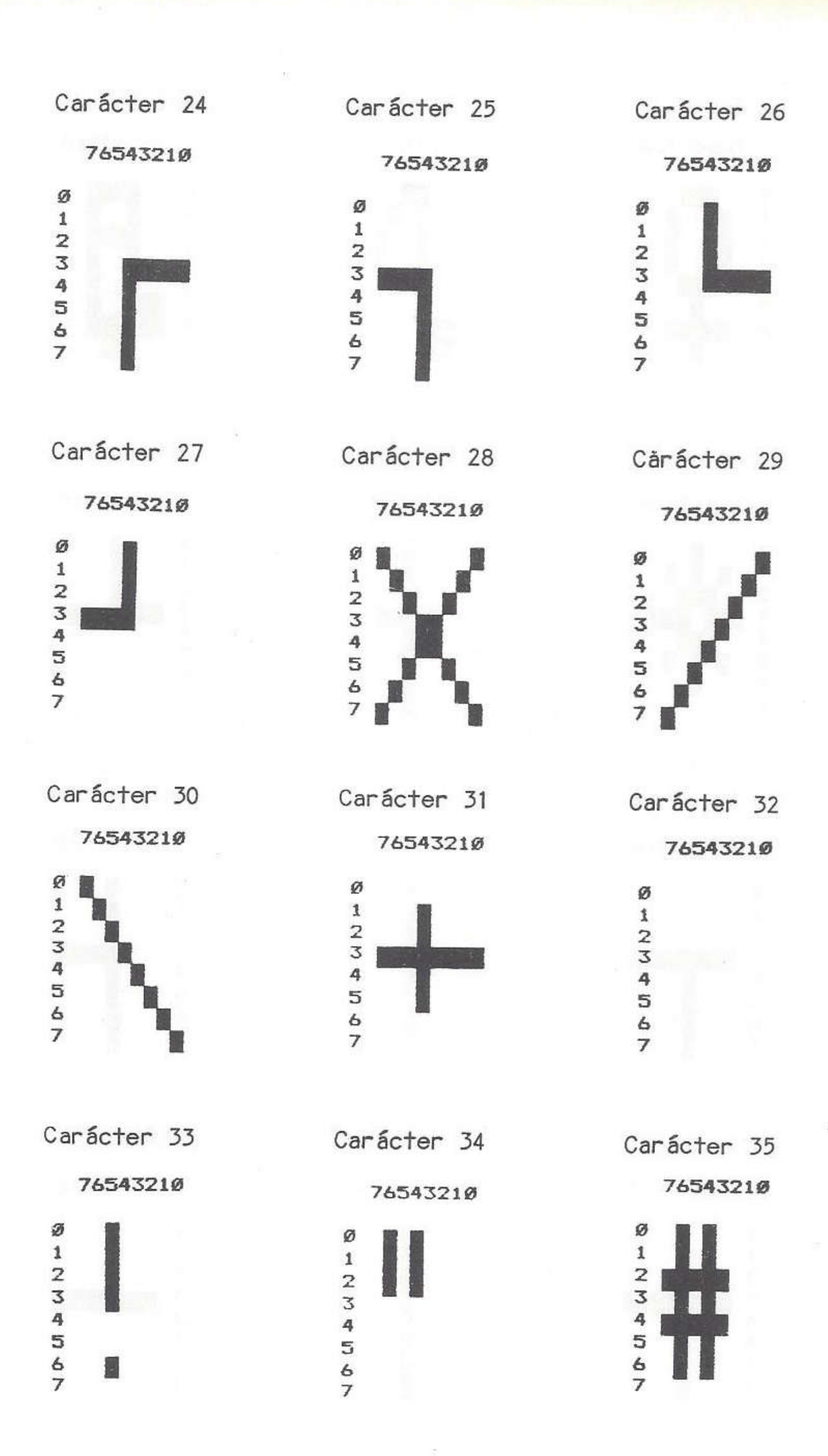
Posibles valores que podemos entrar en el registro 3 de VDP

parte 2

80	2000	81	2040	82	2080	83	20C0
84	2100	85	2140	86	2180	87	21C0
88	2200	89	2240	8A	2280	8B	22C0
8C	2300	8D	2340	8E	2380	8F	23C0
90	2400	91	2440	92	2480	93	24C0
94	2500	95	2540	96	2580	97	25C0
98	2600	99	2640	9A	2680	9B	26C0
9C	2700	9D	2740	9E	2780	9F	27C0
A0	2800	A1	2840	A2	2880	A3	28C0
A4	2900	A5	2940	A6	2980	A7	29C0
A8	2A00	A9	2A40	AA	2A80	AB	2AC0
AC	2B00	AD	2B40	AE	2B80	AF	2BC0
B0	2C00	B1	2C40	B2	2C80	B3	2CC0
B4	2D00	B5	2D40	B6	2D80	B7	2DC0
B8	2E00	B9	2E40	BA	2E80	BB	2EC0
BC	2F00	BD	2F40	BE	2F80	BF	2FC0
C0	3000	C1	3040	C2	3080	C3	30C0
C4	3100	C5	3140	C6	3180	C7	31C0
C8	3200	C9	3240	CA	3280	CB	32C0
CC	3300	CD	3340	CE	3380	CF	33C0
D0	3400	D1	3440	D2	3480	D3	34C0
D4	3500	D5	3540	D6	3580	D7	35C0
D8	3600	D9	3640	DA	3680	DB	36C0
DC	3700	DD	3740	DE	3780	DF	37C0
E0	3800	E1	3840	E2	3880	E3	38C0
E4	3900	E5	3940	E6	3980	E7	39C0
E8	3A00	E9	3A40	EA	3A80	EB	3AC0
EC	3B00	ED	3B40	EE	3B80	EF	3BC0
F0	3C00	F1	3C40	F2	3C80	F3	3CC0
F4	3D00	F5	3D40	F6	3D80	F7	3DC0
F8	3E00	F9	3E40	FA	3E80	FB	3EC0
FC	3F00	FD	3F40	FE	3F80	FF	3FC0


Carácter 0
76543210
Ø
1
2
3
4
5
6
7
Carácter 1
76543210
Ø
1
2
3
4
5
6
7
Carácter 2
76543210
Ø
1
2
3
4
5
6
7
Carácter 3
76543210
Ø
1
2
3
4
5
6
7
Carácter 4
76543210
Ø
1
2
3
4
5
6
7
Carácter 5
76543210
Ø
1
2
3
4
5
6
7
Carácter 6
76543210
Ø
1
2
3
4
5
6
7
Carácter 7
76543210
Ø
1
2
3
4
5
6
7
Carácter 8
76543210
Ø
1
2
3
4
5
6
7
Carácter 9
76543210
Ø
1
2
3
4
5
6
7
Carácter 10
76543210
Ø
1
2
3
4
5
6
7
Carácter 11
76543210
Ø
1
2
3
4
5
6
7


Carácter 12
76543210
0
1
2
3
4
5
6
7
Carácter 13
76543210
0
1
2
3
4
5
6
7
Carácter 14
76543210
0
1
2
3
4
5
6
7
Carácter 15
76543210
0
1
2
3
4
5
6
7
Carácter 16
76543210
0
1
2
3
4
5
6
7
Carácter 17
76543210
0
1
2
3
4
5
6
7
Carácter 18
76543210
0
1
2
3
4
5
6
7
Carácter 19
76543210
0
1
2
3
4
5
6
7
Carácter 20
76543210
0
1
2
3
4
5
6
7
Carácter 21
76543210
0
1
2
3
4
5
6
7
Carácter 22
76543210
0
1
2
3
4
5
6
7
Carácter 23
76543210
0
1
2
3
4
5
6
7

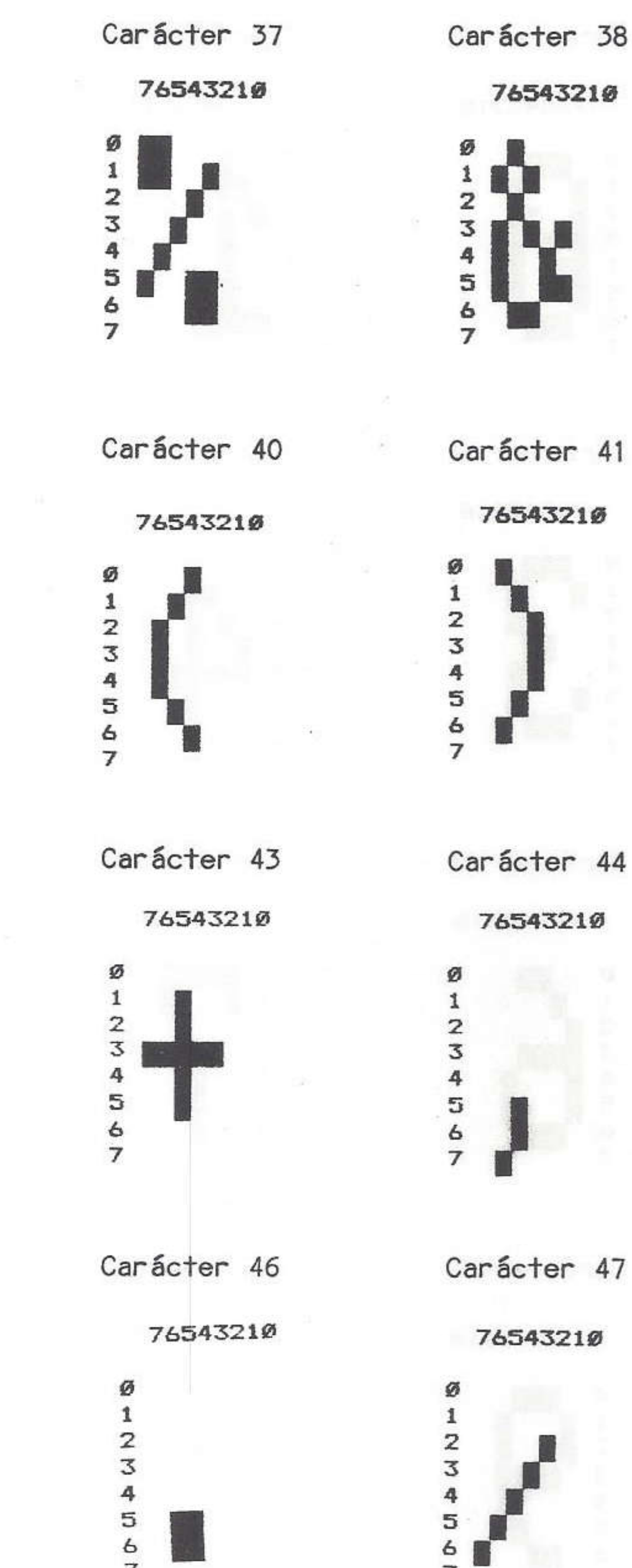

Carácter 24
76543210
0
1
2
3
4
5
6
7
Carácter 25
76543210
0
1
2
3
4
5
6
7
Carácter 26
76543210
0
1
2
3
4
5
6
7
Carácter 27
76543210
0
1
2
3
4
5
6
7
Carácter 28
76543210
0
1
2
3
4
5
6
7
Carácter 29
76543210
0
1
2
3
4
5
6
7
Carácter 30
76543210
0
1
2
3
4
5
6
7
Carácter 31
76543210
0
1
2
3
4
5
6
7
Carácter 32
76543210
0
1
2
3
4
5
6
7
Carácter 33
76543210
0
1
2
3
4
5
6
7
Carácter 34
76543210
0
1
2
3
4
5
6
7
Carácter 35
76543210
0
1
2
3
4
5
6
7

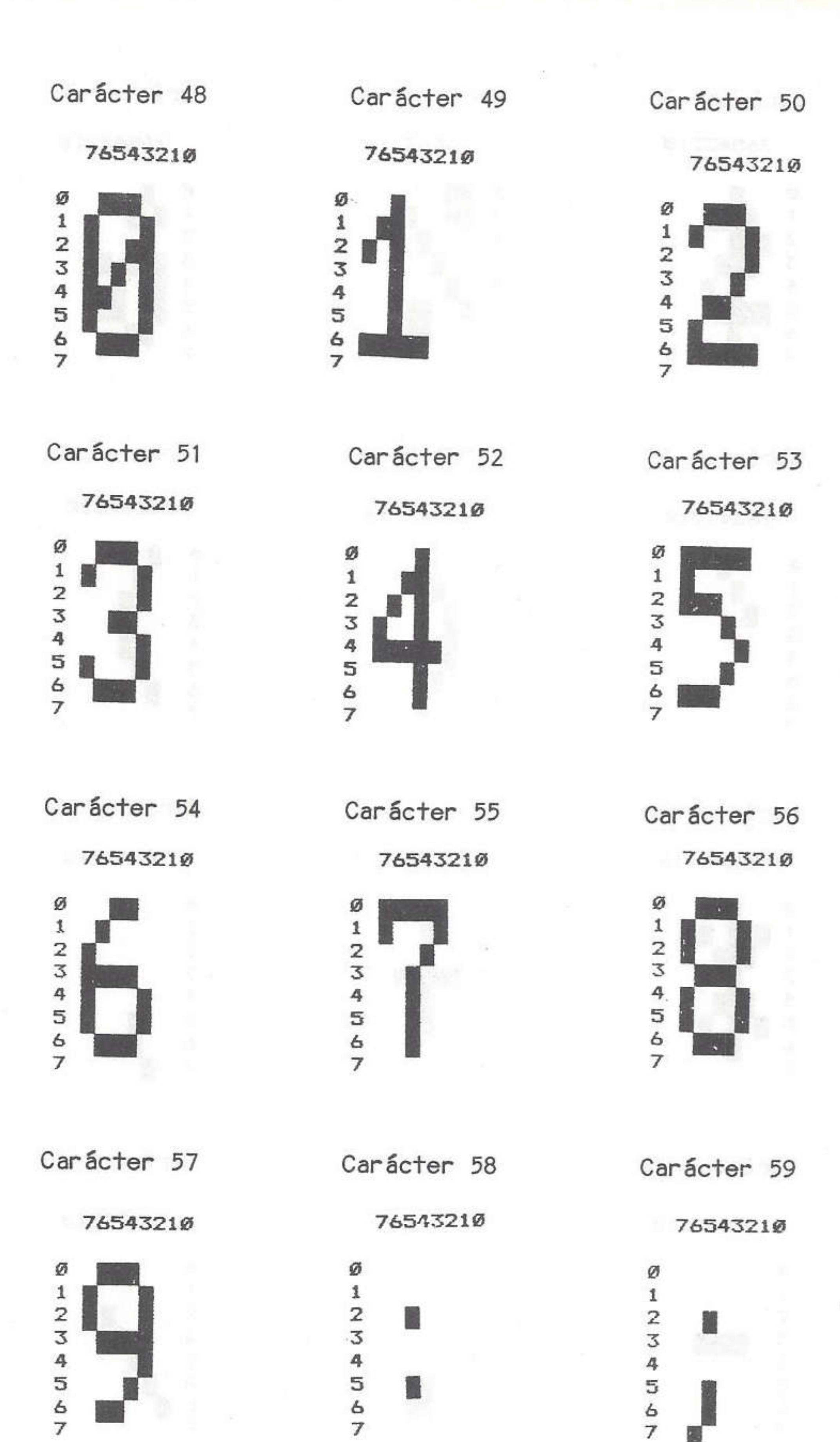
Carácter 36

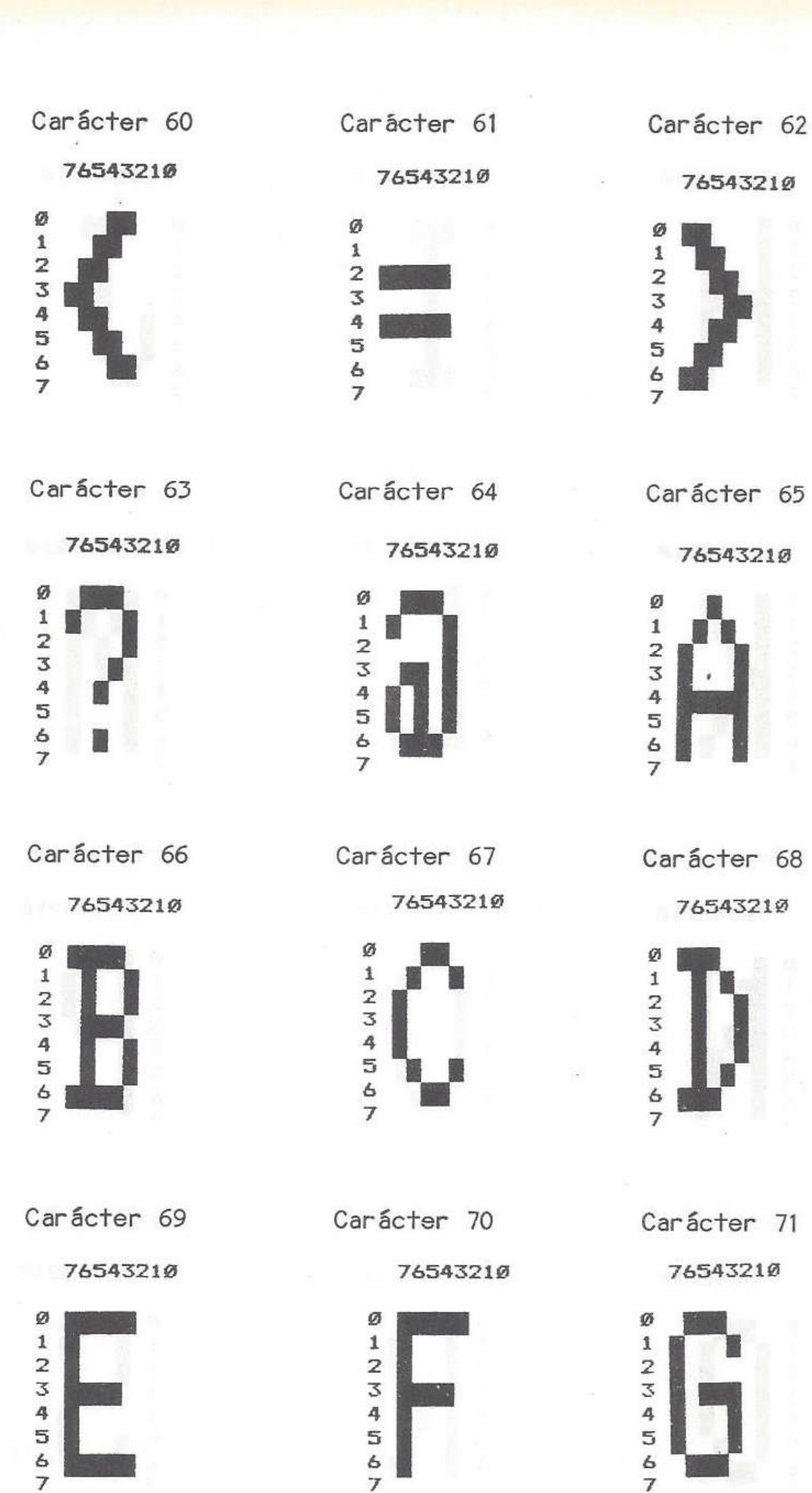
Carácter 39

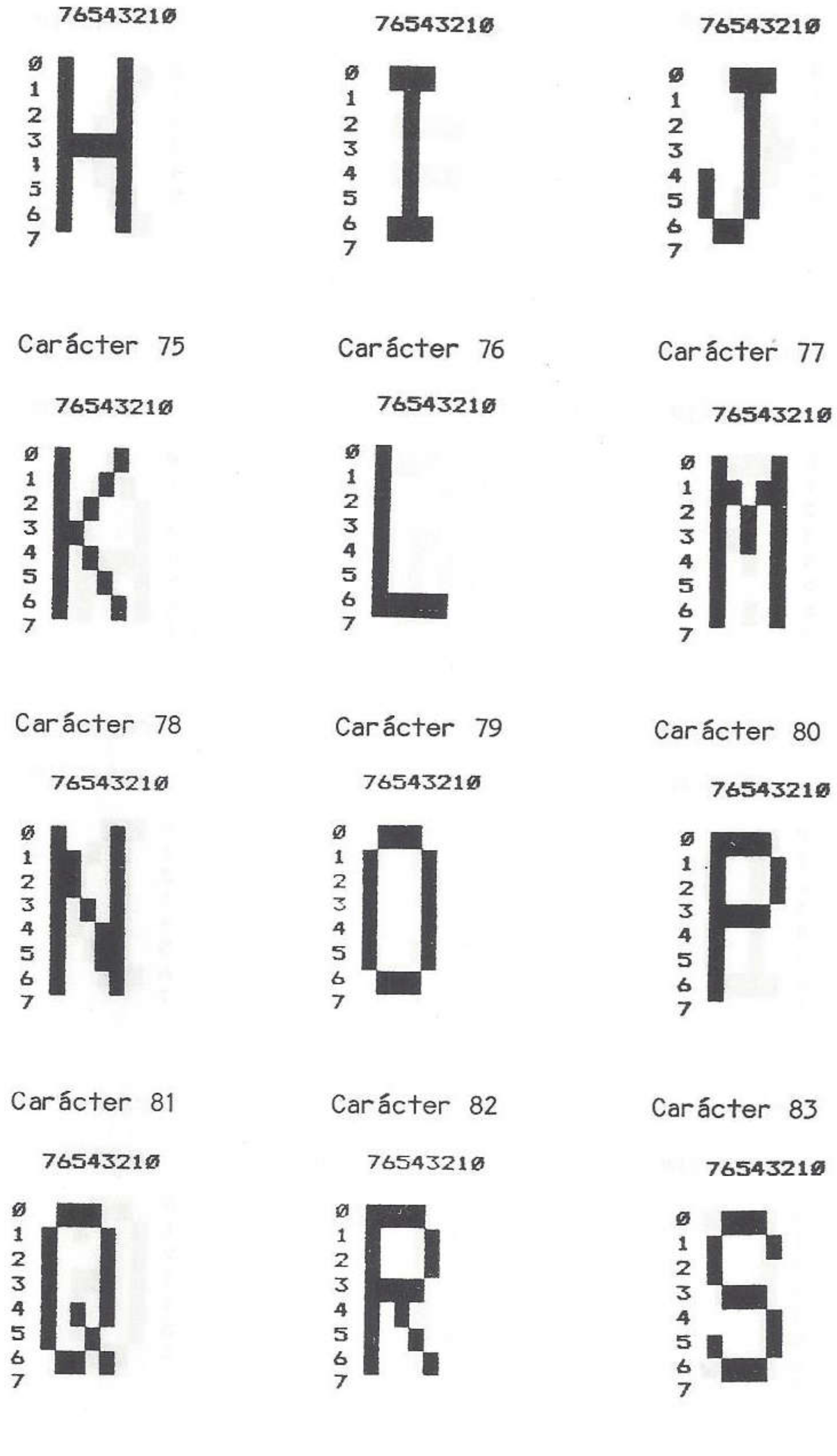
Carácter 42

Carácter 45




Carácter 48
76543210
Carácter 49
76543210
Carácter 50
76543210
Carácter 51
76543210
Carácter 52
76543210
Carácter 53
76543210
Carácter 54
76543210
Carácter 55
76543210
Carácter 56
76543210
Carácter 57
76543210
Carácter 58
76543210
Carácter 59
76543210


Carácter 60
76543210
0
1
2
3
4
5
6
7
Carácter 61
76543210
0
1
2
3
4
5
6
7
Carácter 62
76543210
0
1
2
3
4
5
6
7
Carácter 63
76543210
0
1
2
3
4
5
6
7
Carácter 64
76543210
0
1
2
3
4
5
6
7
Carácter 65
76543210
0
1
2
3
4
5
6
7
Carácter 66
76543210
0
1
2
3
4
5
6
7
Carácter 67
76543210
0
1
2
3
4
5
6
7
Carácter 68
76543210
0
1
2
3
4
5
6
7
Carácter 69
76543210
0
1
2
3
4
5
6
7
Carácter 70
76543210
0
1
2
3
4
5
6
7
Carácter 71
76543210
0
1
2
3
4
5
6
7


Carácter 72
76543210
Ø
1
2
3
4
5
6
7
Carácter 73
76543210
Ø
1
2
3
4
5
6
7
Carácter 74
76543210
Ø
1
2
3
4
5
6
7
Carácter 75
76543210
Carácter 76
76543210
Carácter 77
76543210
Carácter 78
76543210
Carácter 79
76543210
Carácter 80
76543210
Carácter 81
76543210
Carácter 82
76543210
Carácter 83
76543210

Carácter 84

76543210

Ø 1 2 3 4 5 6 7

Carácter 85

76543210

Ø 1 2 3 4 5 6 7

Carácter 86

76543210

Ø 1 2 3 4 5 6 7

Carácter 87

76543210

Ø 1 2 3 4 5 6 7

Carácter 88

76543210

Ø 1 2 3 4 5 6 7

Carácter 89

76543210

Ø 1 2 3 4 5 6 7

Carácter 90

76543210

Ø 1 2 3 4 5 6 7

Carácter 91

76543210

Ø 1 2 3 4 5 6 7

Carácter 92



Carácter 93

76543210

Ø 1 2 3 4 5 6 7

Carácter 94

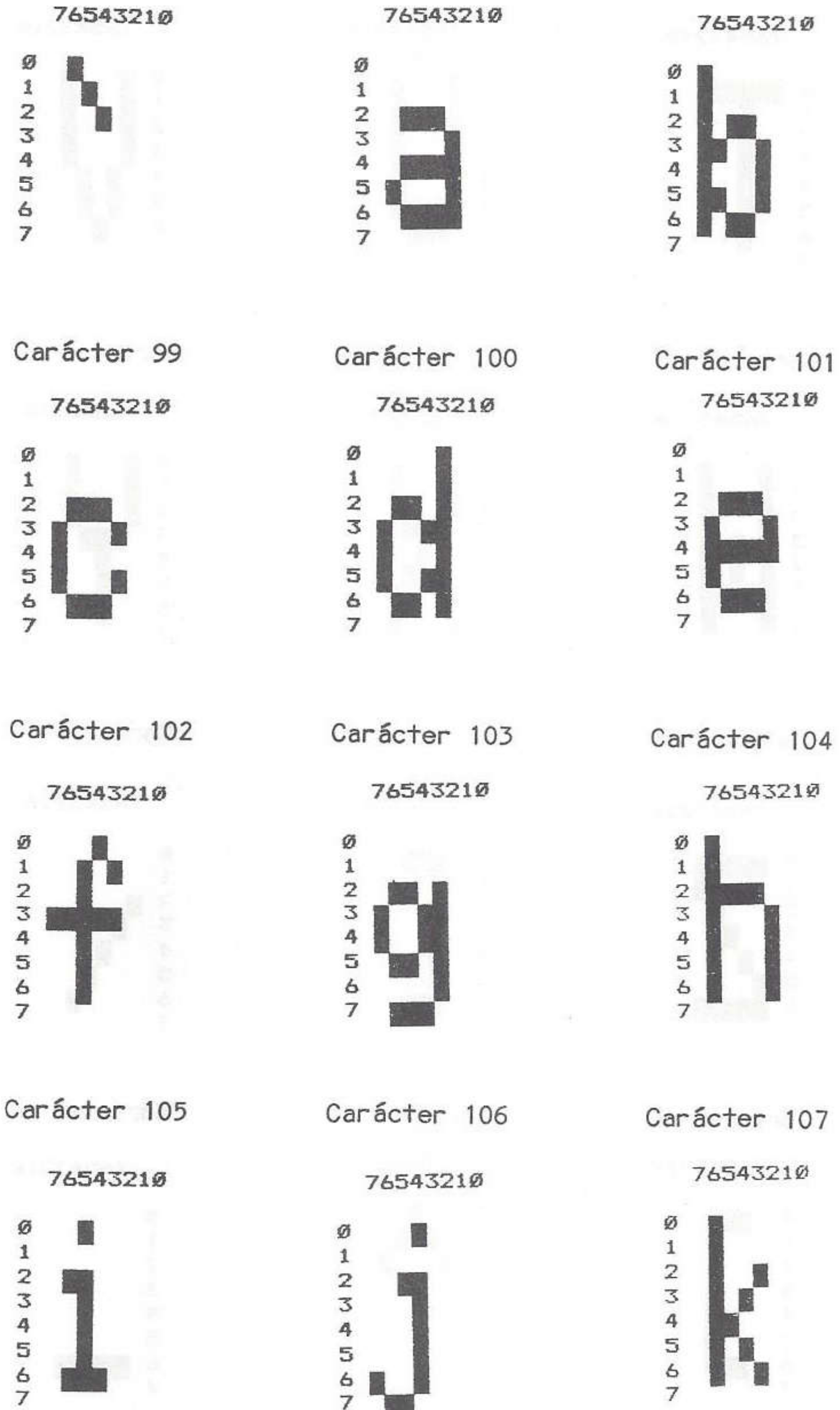
76543210

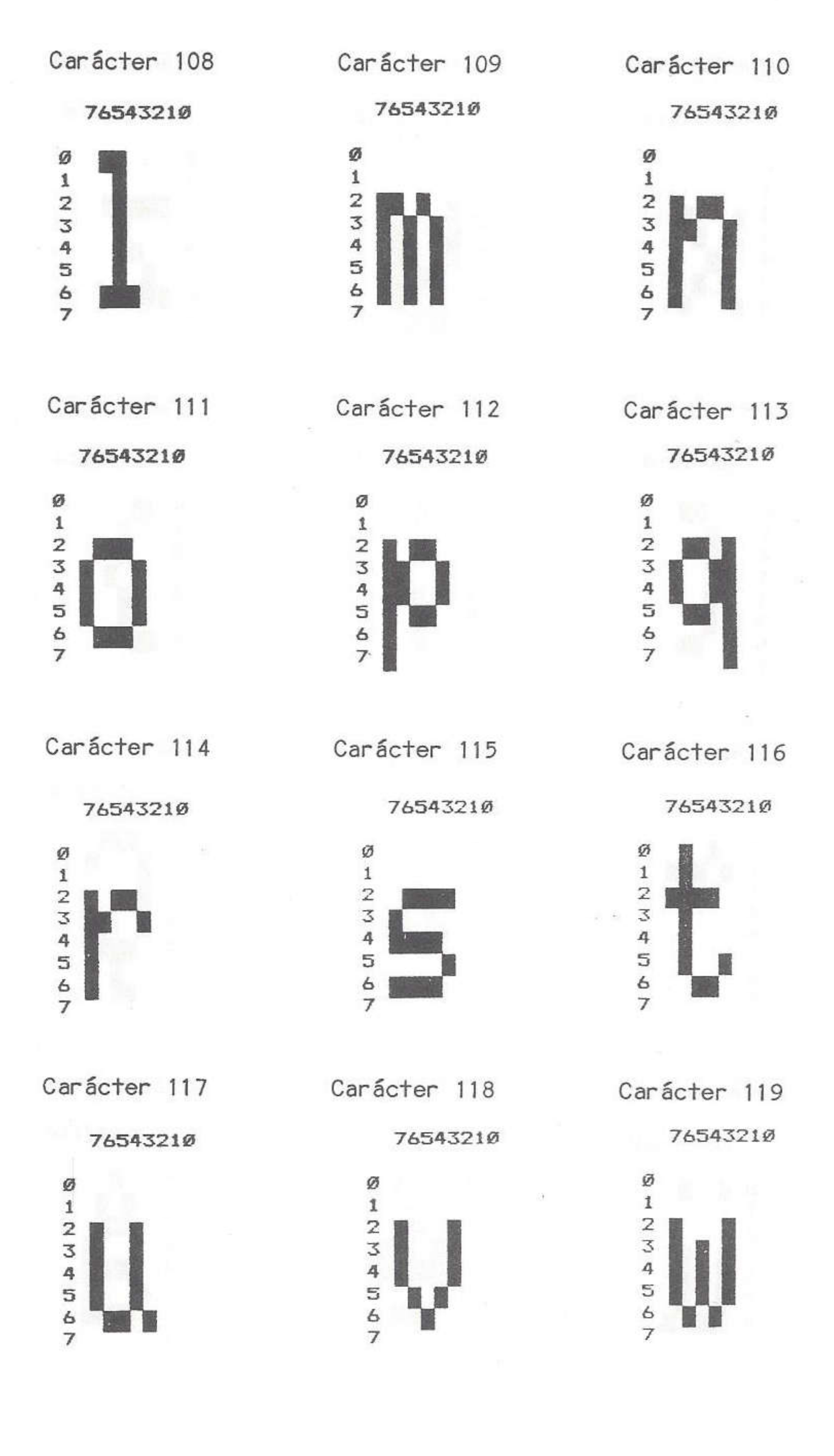
Ø 1 2 3 4 5 6 7

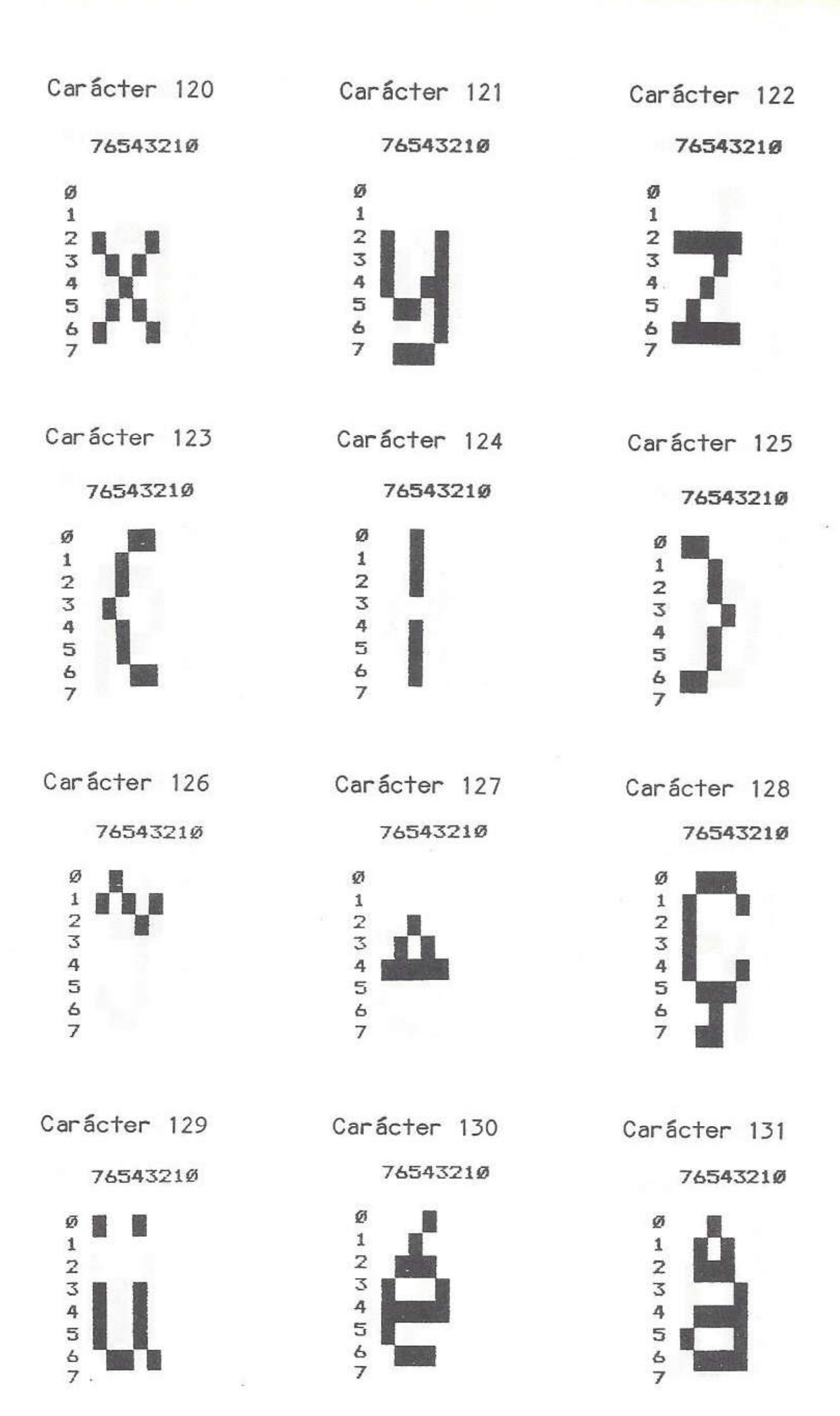
Carácter 95

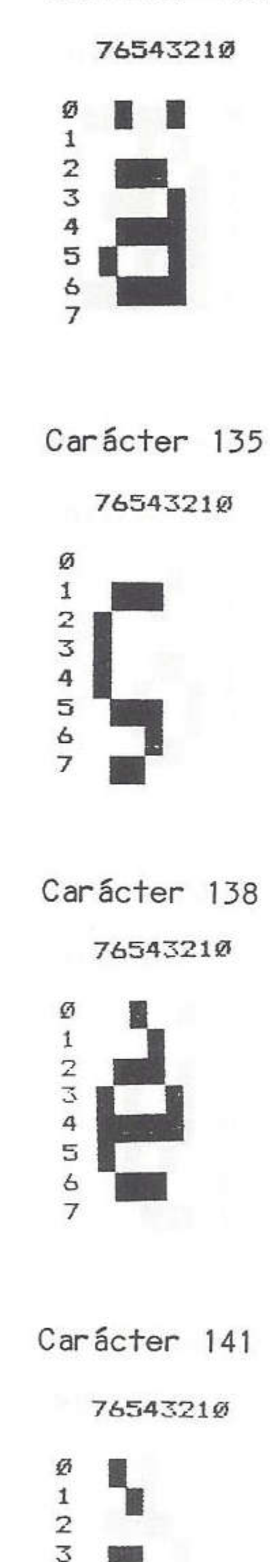
76543210

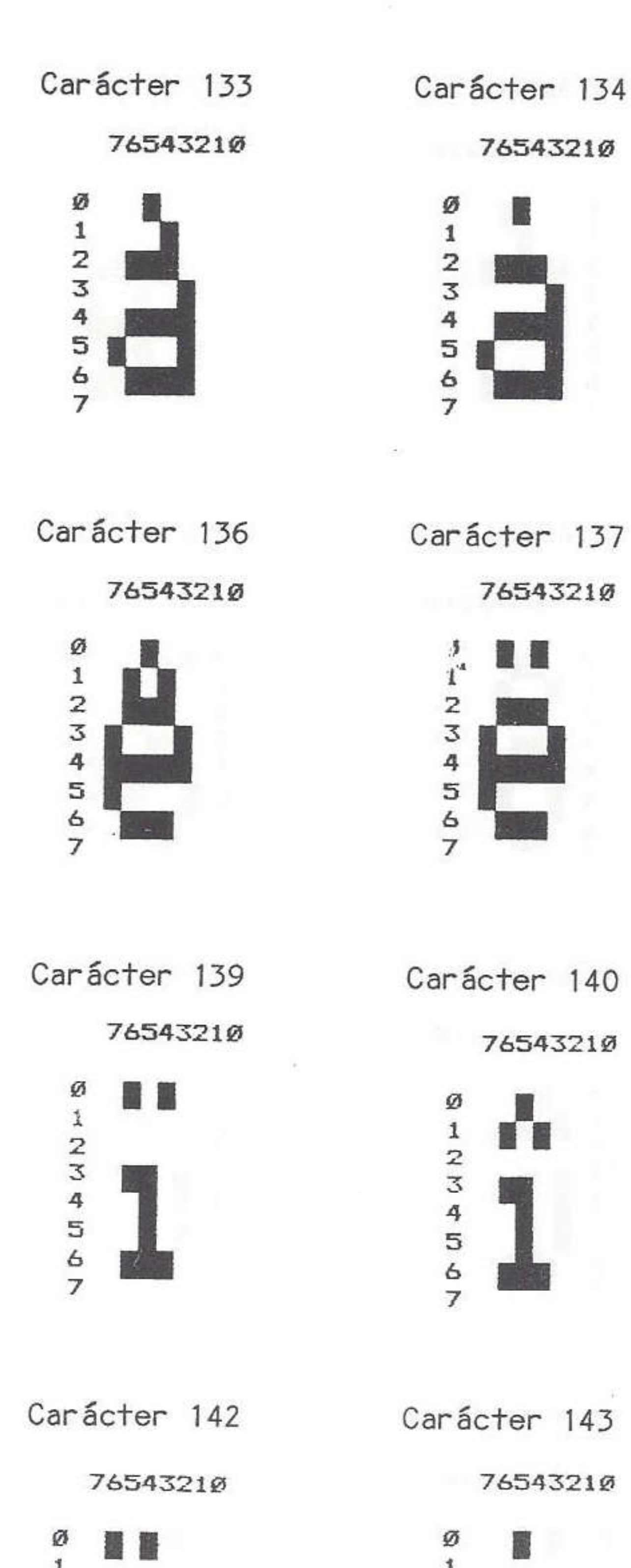
Ø 1 2 3 4 5 6 7

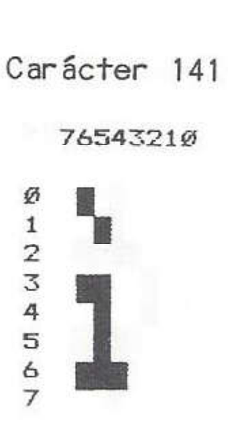

Carácter 96
76543210
Ø
1
2
3
4
5
6
7
Carácter 97
76543210
Ø
1
2
3
4
5
6
7
Carácter 98
76543210
Ø
1
2
3
4
5
6
7
Carácter 99
76543210
Ø
1
2
3
4
5
6
7
Carácter 100
76543210
Ø
1
2
3
4
5
6
7
Carácter 101
76543210
Ø
1
2
3
4
5
6
7
Carácter 102
76543210
Ø
1
2
3
4
5
6
7
Carácter 103
76543210
Ø
1
2
3
4
5
6
7
Carácter 104
76543210
Ø
1
2
3
4
5
6
7
Carácter 105
76543210
Ø
1
2
3
4
5
6
7
Carácter 106
76543210
Ø
1
2
3
4
5
6
7
Carácter 107
76543210
Ø
1
2
3
4
5
6
7

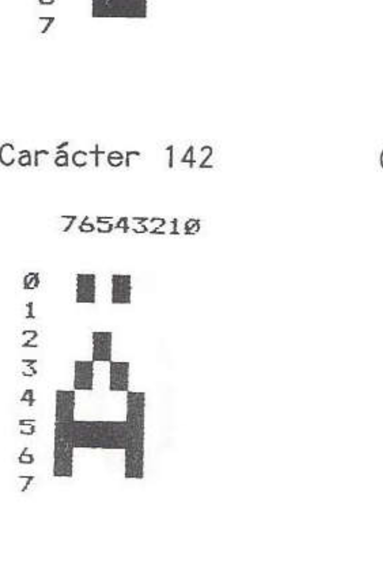

Carácter 108
76543210
Ø
1
2
3
4
5
6
7
Carácter 109
76543210
Ø
1
2
3
4
5
6
7
Carácter 110
76543210
Ø
1
2
3
4
5
6
7
Carácter 111
76543210
Ø
1
2
3
4
5
6
7
Carácter 112
76543210
Ø
1
2
3
4
5
6
7
Carácter 113
76543210
Ø
1
2
3
4
5
6
7
Carácter 114
76543210
Ø
1
2
3
4
5
6
7
Carácter 115
76543210
Ø
1
2
3
4
5
6
7
Carácter 116
76543210
Ø
1
2
3
4
5
6
7
Carácter 117
76543210
Ø
1
2
3
4
5
6
7
Carácter 118
76543210
Ø
1
2
3
4
5
6
7
Carácter 119
76543210
Ø
1
2
3
4
5
6
7


Carácter 120
76543210
Ø 1 2 3 4 5 6 7
Carácter 121
76543210
Ø 1 2 3 4 5 6 7
Carácter 122
76543210
Ø 1 2 3 4 5 6 7
Carácter 123
76543210
Ø 1 2 3 4 5 6 7
Carácter 124
76543210
Ø 1 2 3 4 5 6 7
Carácter 125
76543210
Ø 1 2 3 4 5 6 7
Carácter 126
76543210
Ø 1 2 3 4 5 6 7
Carácter 127
76543210
Ø 1 2 3 4 5 6 7
Carácter 128
76543210
Ø 1 2 3 4 5 6 7
Carácter 129
76543210
Ø 1 2 3 4 5 6 7
Carácter 130
76543210
Ø 1 2 3 4 5 6 7
Carácter 131
76543210
Ø 1 2 3 4 5 6 7









Carácter 143

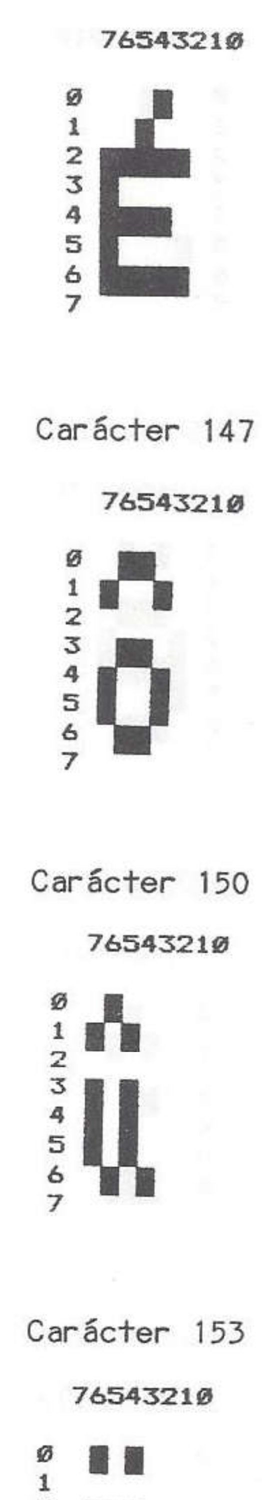
7654321Ø

Ø
1
2
3
4
5
6
7

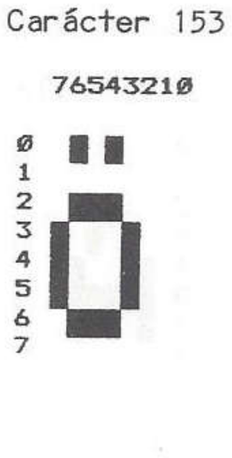
Carácter 144

Carácter 147

Carácter 150



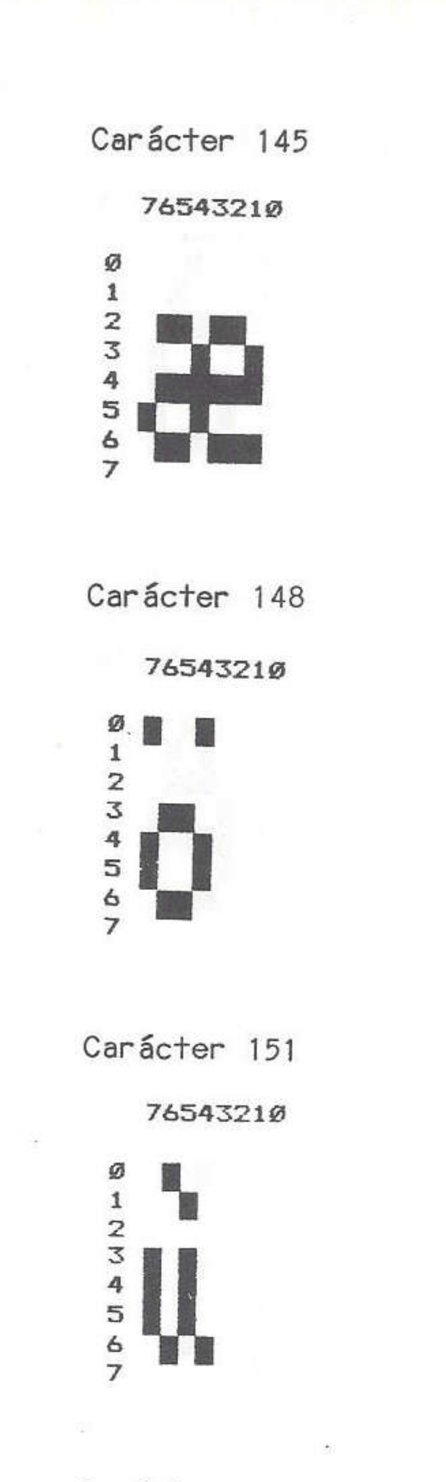
Carácter 153



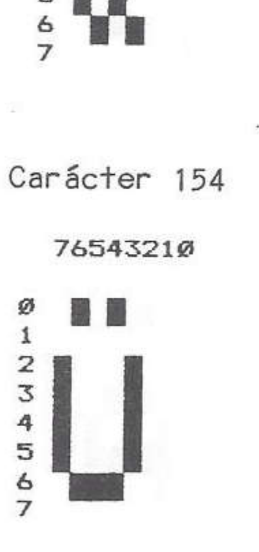
Carácter 145

Carácter 148

Carácter 151



Carácter 154

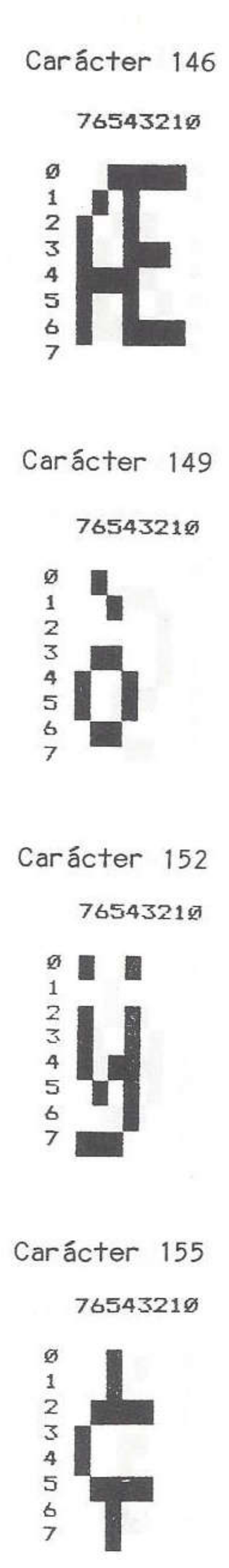


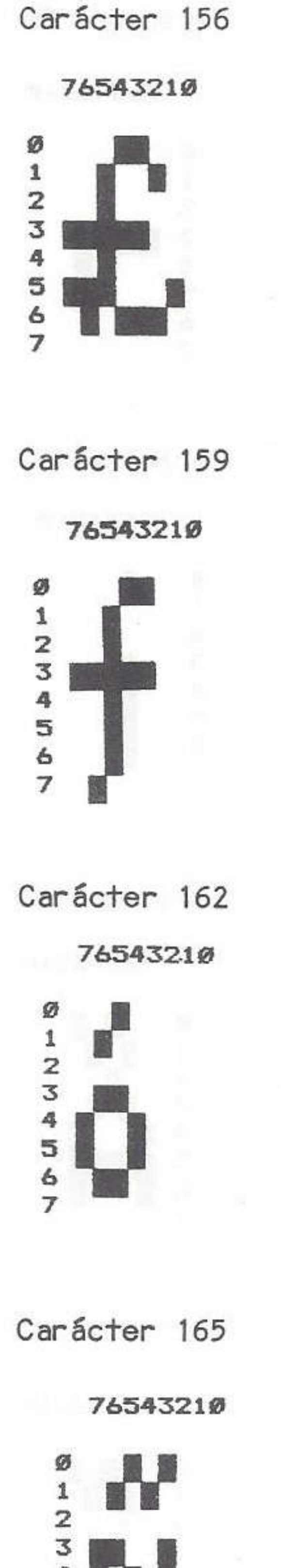
Carácter 146

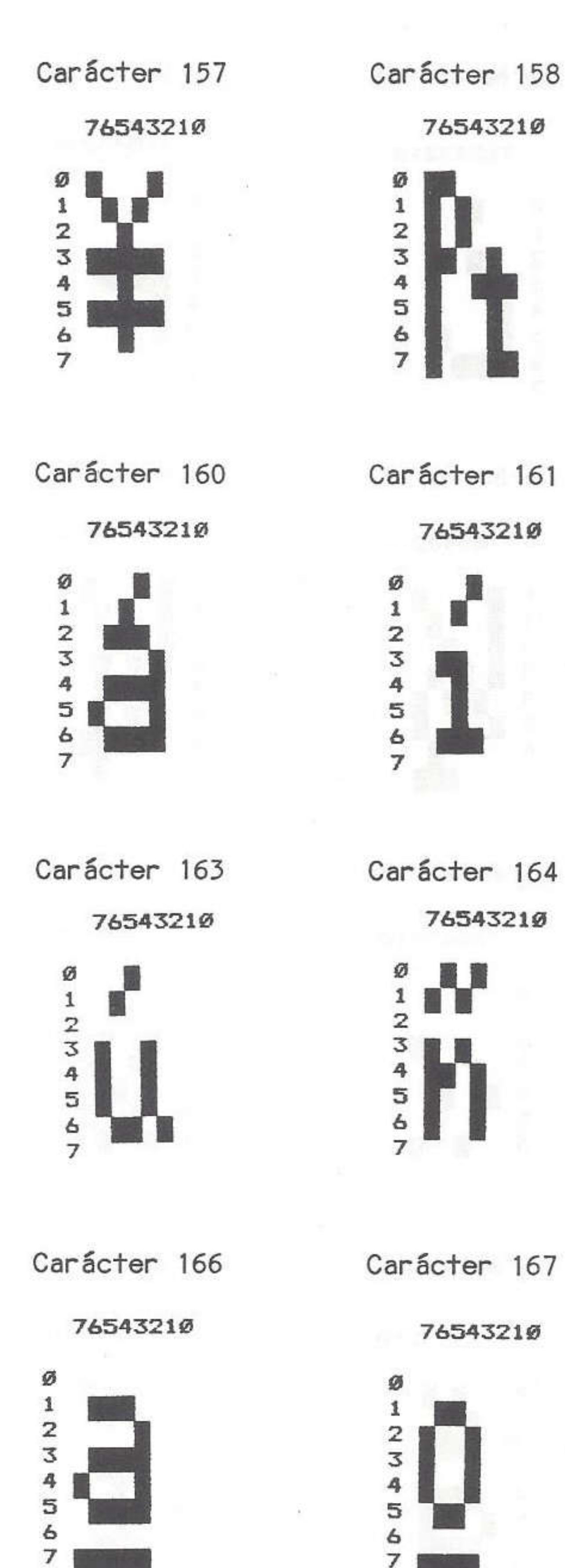
Carácter 149

Carácter 152

Carácter 155







Carácter 156

76543210

Carácter 157

76543210

Carácter 158

76543210

Carácter 159

76543210

Carácter 160

76543210

Carácter 161

76543210



Carácter 162

76543210

Carácter 163

76543210

Carácter 164

76543210

Carácter 165

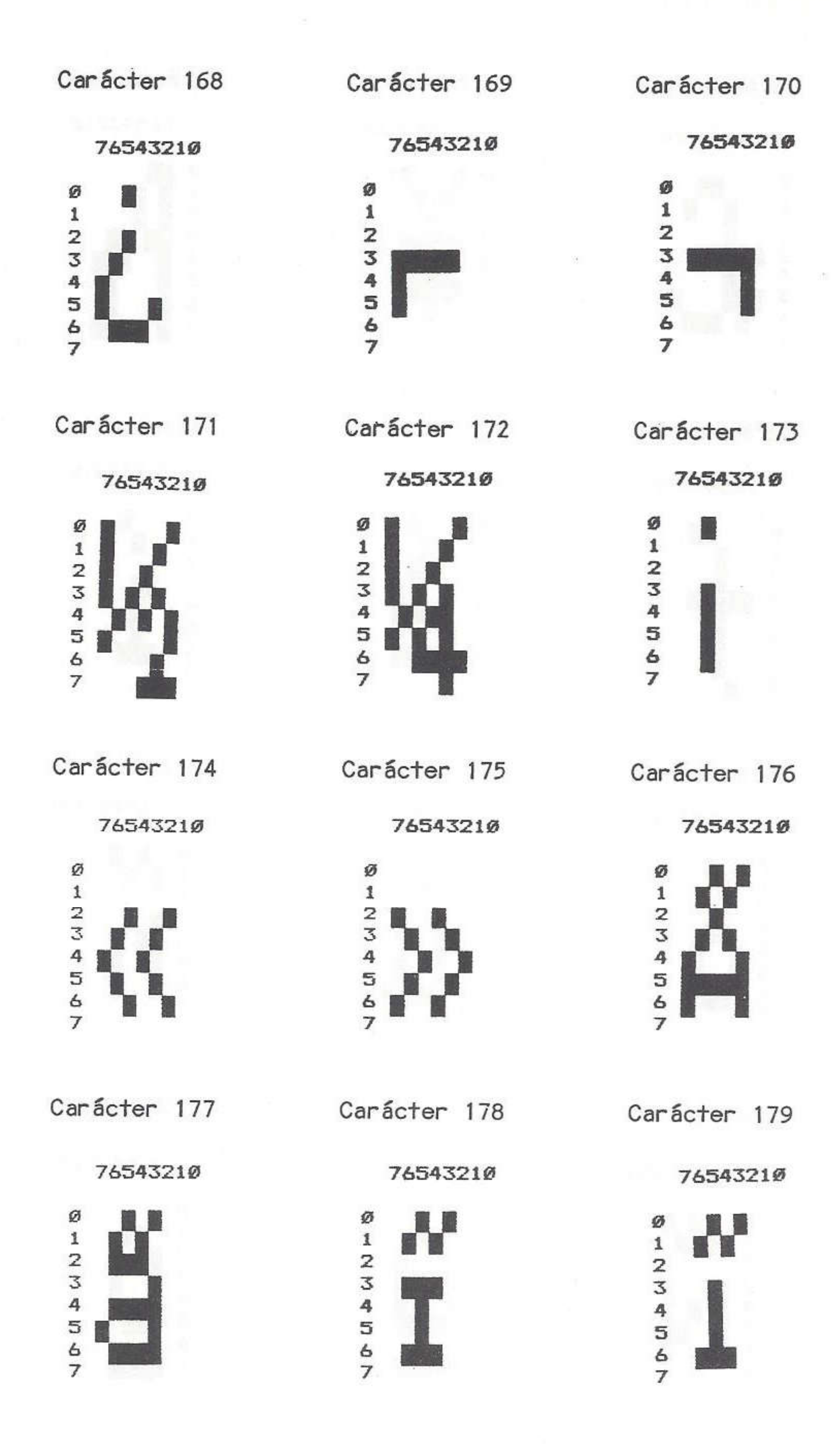
76543210

Carácter 166

76543210

Carácter 167

76543210


Carácter 168
76543210
Ø
1
2
3
4
5
6
7
Carácter 169
76543210
Ø
1
2
3
4
5
6
7
Carácter 170
76543210
Ø
1
2
3
4
5
6
7
Carácter 171
76543210
Ø
1
2
3
4
5
6
7
Carácter 172
76543210
Ø
1
2
3
4
5
6
7
Carácter 173
76543210
Ø
1
2
3
4
5
6
7
Carácter 174
76543210
Ø
1
2
3
4
5
6
7
Carácter 175
76543210
Ø
1
2
3
4
5
6
7
Carácter 176
76543210
Ø
1
2
3
4
5
6
7
Carácter 177
76543210
Ø
1
2
3
4
5
6
7
Carácter 178
76543210
Ø
1
2
3
4
5
6
7
Carácter 179
76543210
Ø
1
2
3
4
5
6
7

Carácter 180

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 181

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 182

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 183

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 184

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 185

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 186

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 187

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 188

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 189

7654321Ø

Ø
1
2
3
4
5
6
7

Carácter 190

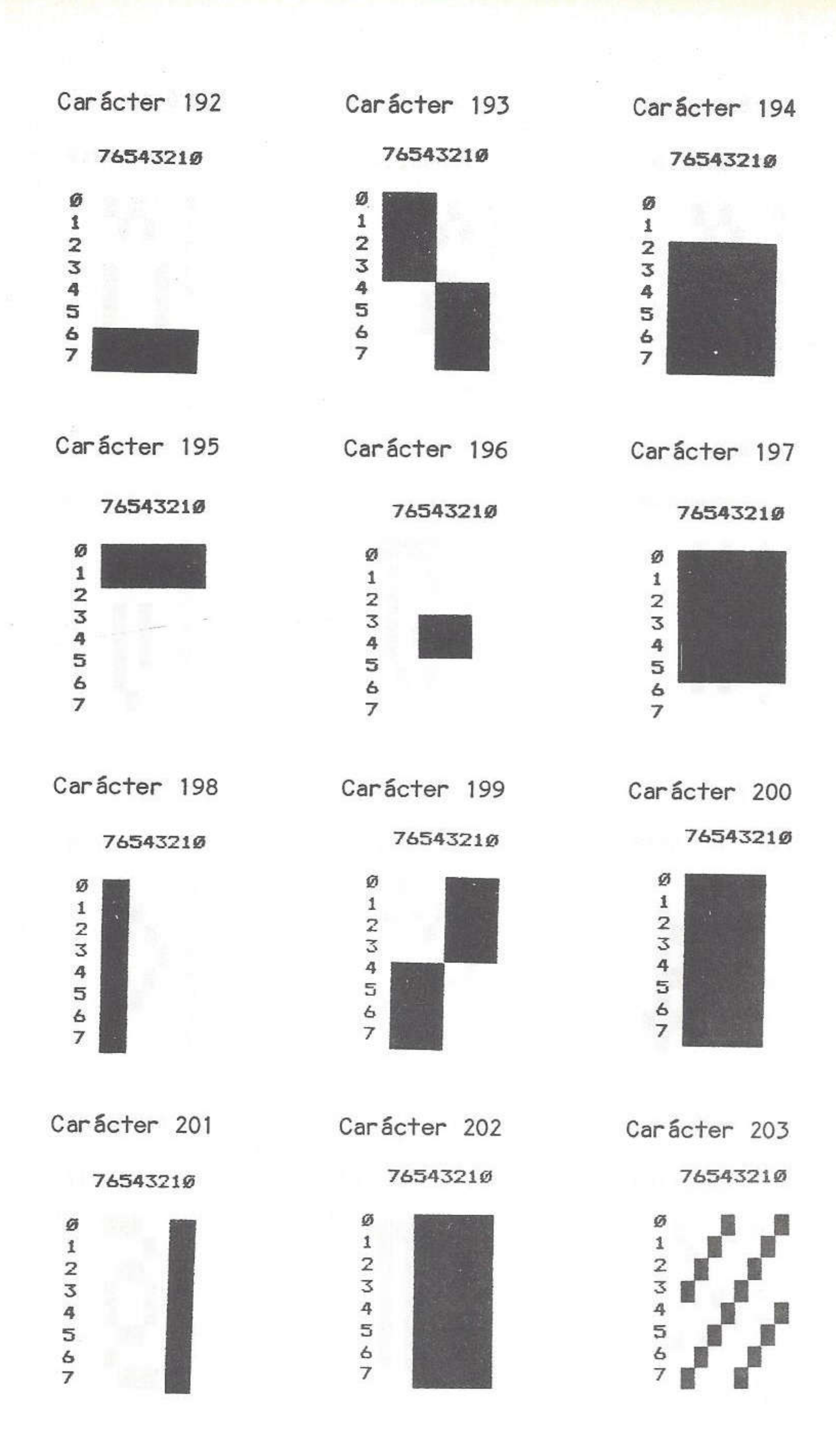
7654321Ø

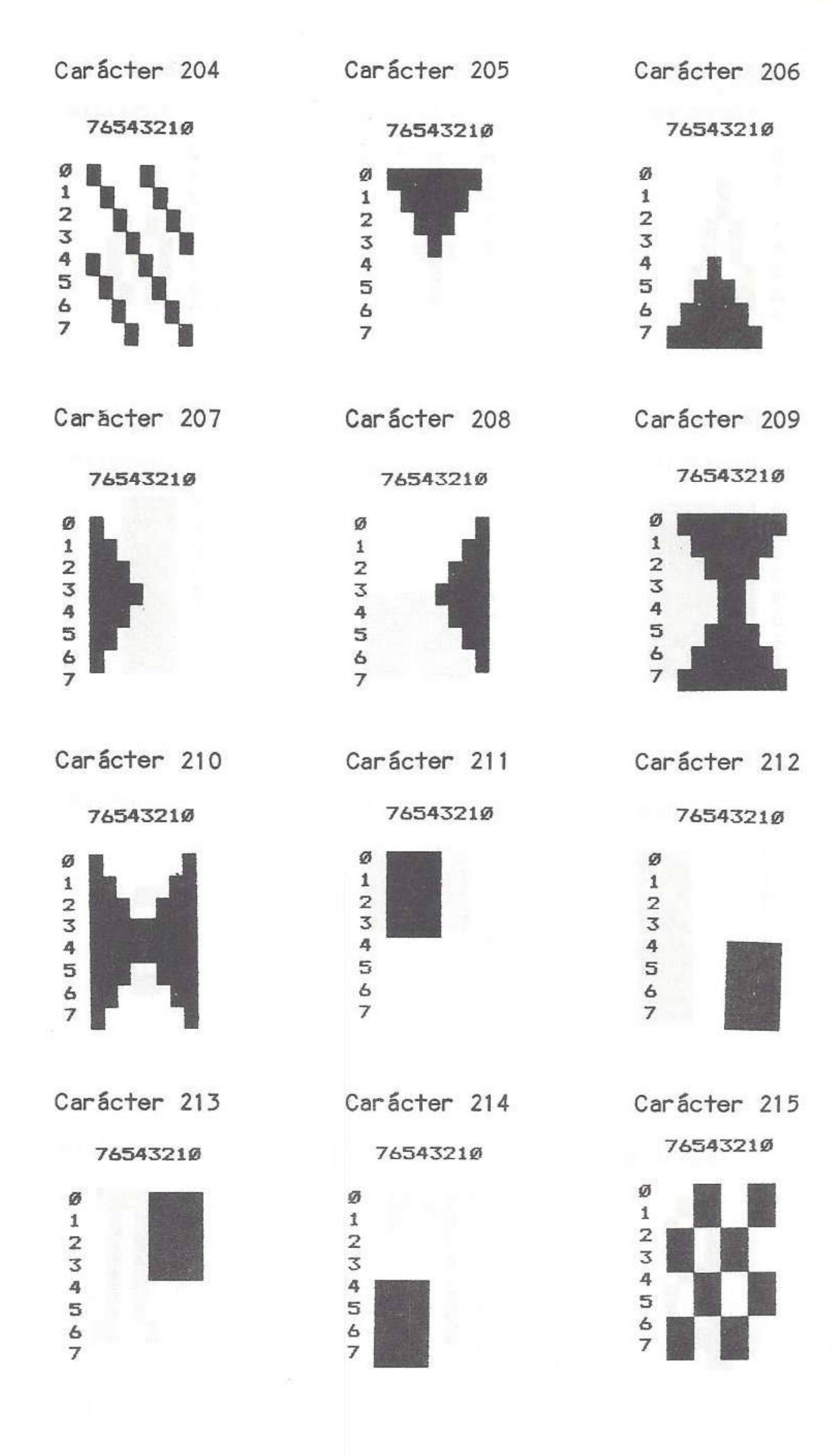
Ø
1
2
3
4
5
6
7

Carácter 191

7654321Ø

Ø
1
2
3
4
5
6
7


Carácter 192
76543210
0
1
2
3
4
5
6
7
Carácter 193
76543210
0
1
2
3
4
5
6
7
Carácter 194
76543210
0
1
2
3
4
5
6
7
Carácter 195
76543210
0
1
2
3
4
5
6
7
Carácter 196
76543210
0
1
2
3
4
5
6
7
Carácter 197
76543210
0
1
2
3
4
5
6
7
Carácter 198
76543210
0
1
2
3
4
5
6
7
Carácter 199
76543210
0
1
2
3
4
5
6
7
Carácter 200
76543210
0
1
2
3
4
5
6
7
Carácter 201
76543210
0
1
2
3
4
5
6
7
Carácter 202
76543210
0
1
2
3
4
5
6
7
Carácter 203
76543210
0
1
2
3
4
5
6
7



Carácter 204
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 205
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 206
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 207
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 208
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 209
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 210
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 211
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 212
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 213
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 214
7654321Ø
Ø 1 2 3 4 5 6 7
Carácter 215
7654321Ø
Ø 1 2 3 4 5 6 7

Carácter 216

76543210

Ø
1
2
3
4
5
6
7

Carácter 217

76543210

Ø
1
2
3
4
5
6
7

Carácter 218

76543210

Ø
1
2
3
4
5
6
7

Carácter 219

76543210

Ø
1
2
3
4
5
6
7

Carácter 220

76543210

Ø
1
2
3
4
5
6
7

Carácter 221

76543210

Ø
1
2
3
4
5
6
7

Carácter 222

76543210

Ø
1
2
3
4
5
6
7

Carácter 223

76543210

Ø
1
2
3
4
5
6
7

Carácter 224

76543210

Ø
1
2
3
4
5
6
7

Carácter 225

76543210

Ø
1
2
3
4
5
6
7

Carácter 226

76543210

Ø
1
2
3
4
5
6
7

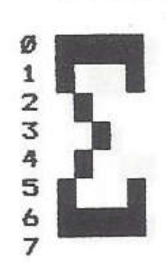
Carácter 227

76543210

Ø
1
2
3
4
5
6
7

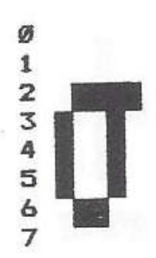
Carácter 228

76543210



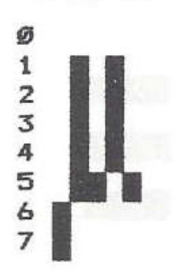
Carácter 229

76543210



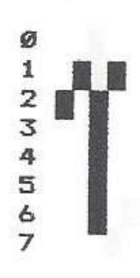
Carácter 230

76543210



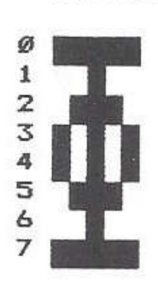
Carácter 231

76543210



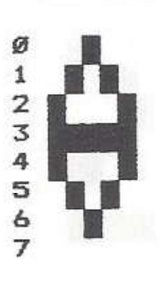
Carácter 232

76543210



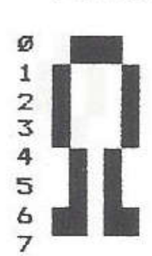
Carácter 233

76543210



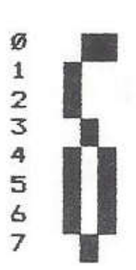
Carácter 234

76543210



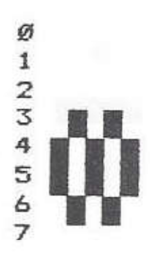
Carácter 235

76543210



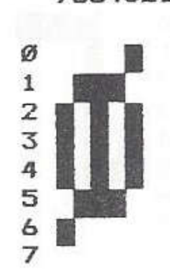
Carácter 236

76543210



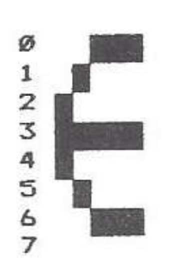
Carácter 237

76543210



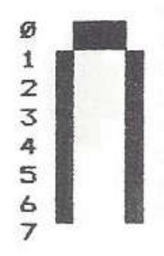
Carácter 238

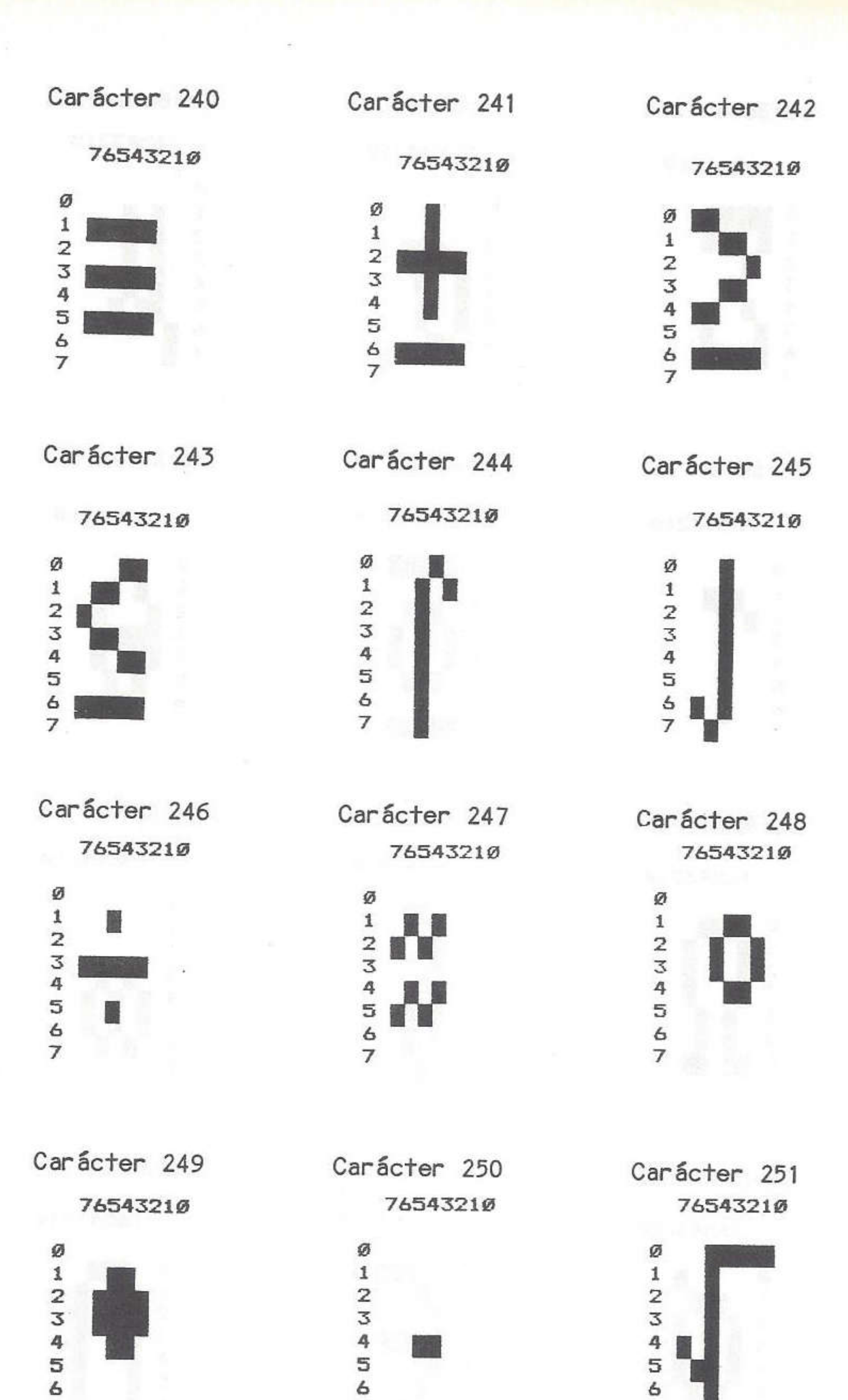
76543210



Carácter 239

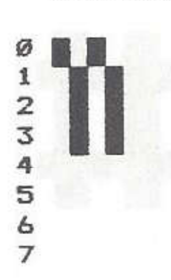
76543210




Carácter 240
76543210
Ø
1
2
3
4
5
6
7
Carácter 241
76543210
Ø
1
2
3
4
5
6
7
Carácter 242
76543210
Ø
1
2
3
4
5
6
7
Carácter 243
76543210
Ø
1
2
3
4
5
6
7
Carácter 244
76543210
Ø
1
2
3
4
5
6
7
Carácter 245
76543210
Ø
1
2
3
4
5
6
7
Carácter 246
76543210
Ø
1
2
3
4
5
6
7
Carácter 247
76543210
Ø
1
2
3
4
5
6
7
Carácter 248
76543210
Ø
1
2
3
4
5
6
7
Carácter 249
76543210
Ø
1
2
3
4
5
6
7
Carácter 250
76543210
Ø
1
2
3
4
5
6
7
Carácter 251
76543210
Ø
1
2
3
4
5
6
7

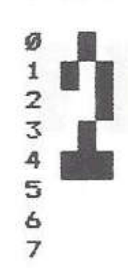
Carácter 252

76543210



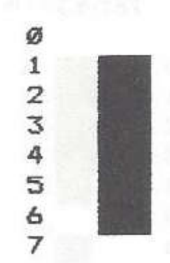
Carácter 253

76543210



Carácter 254

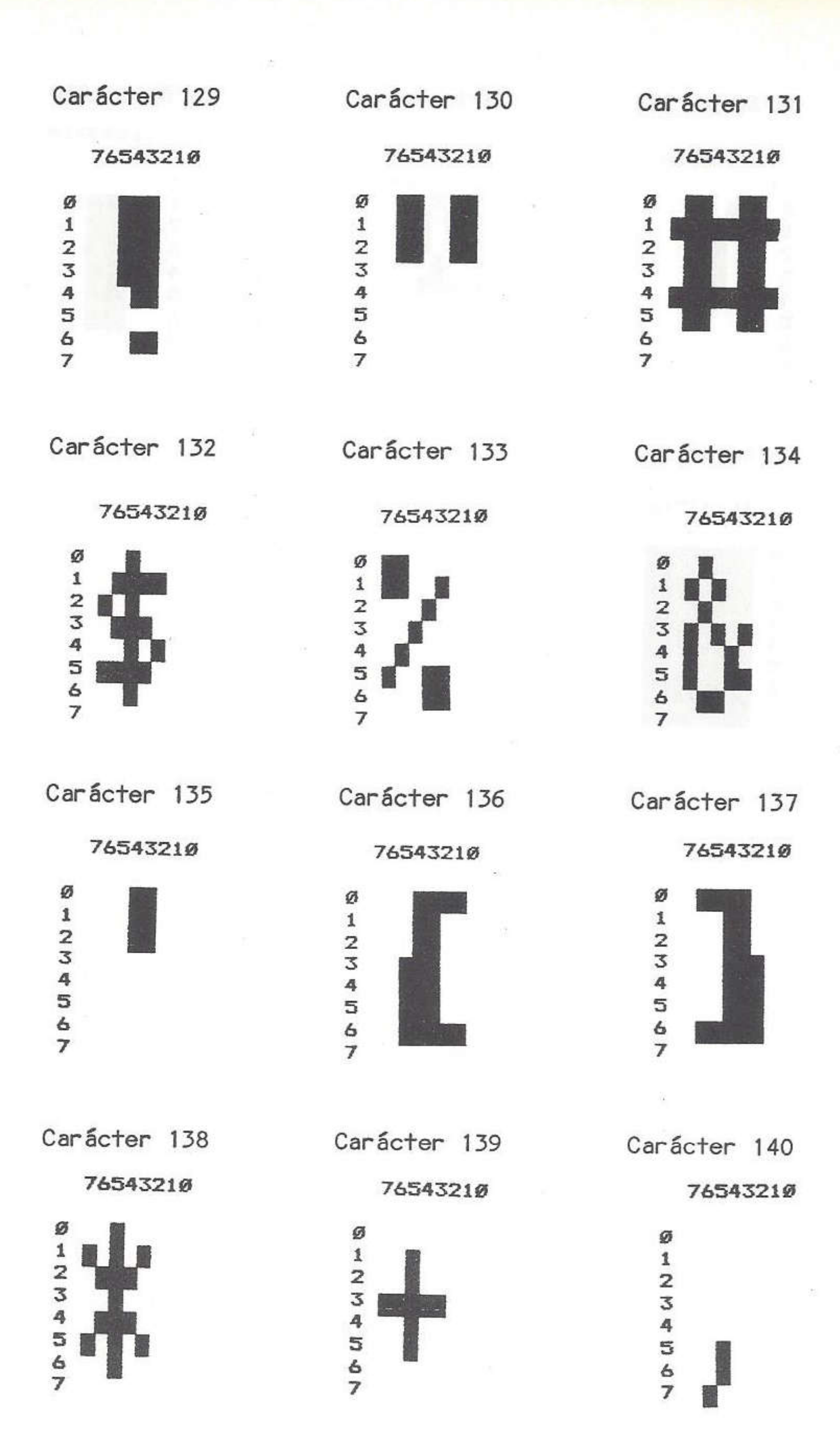
76543210



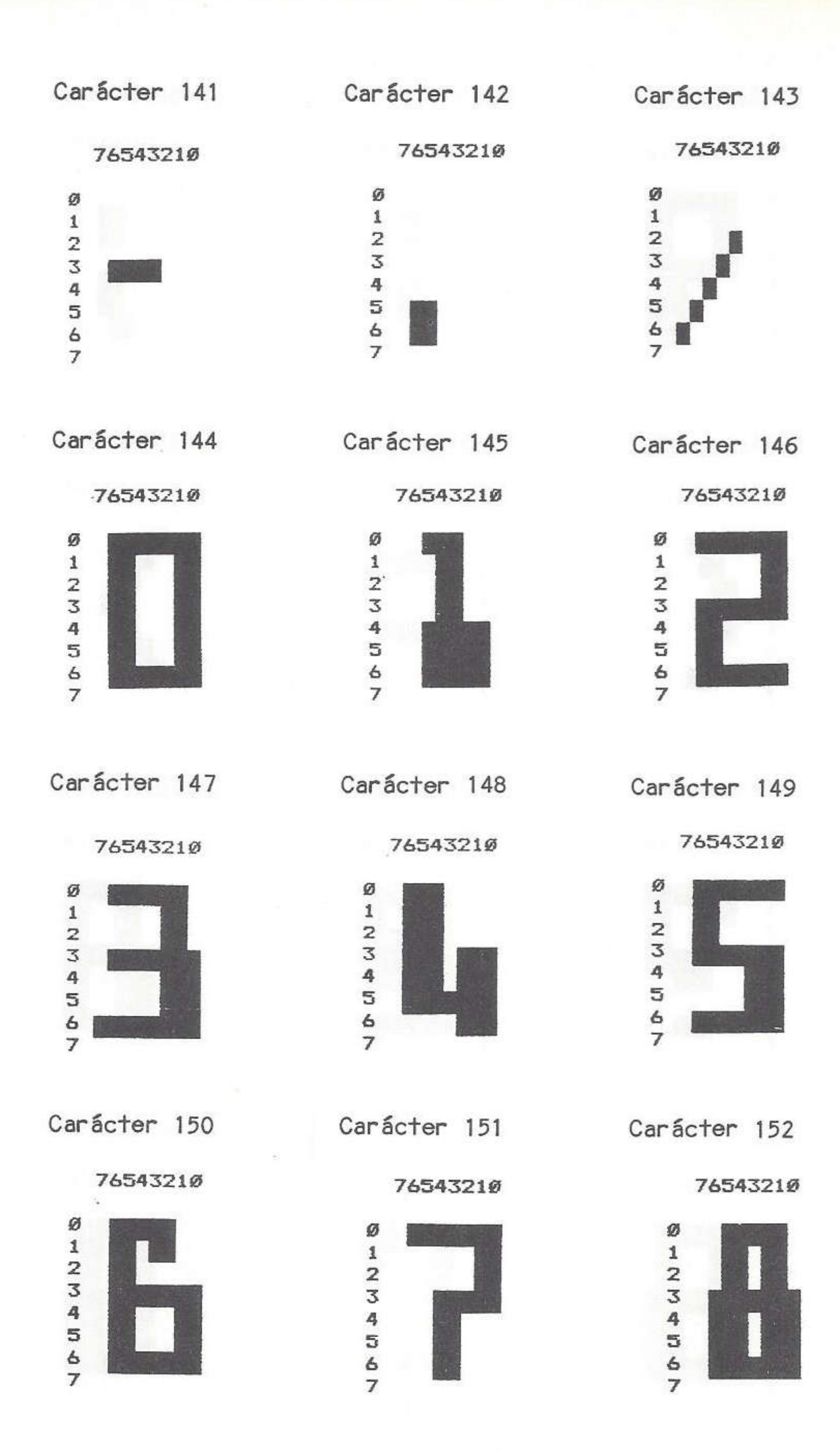
Carácter 255

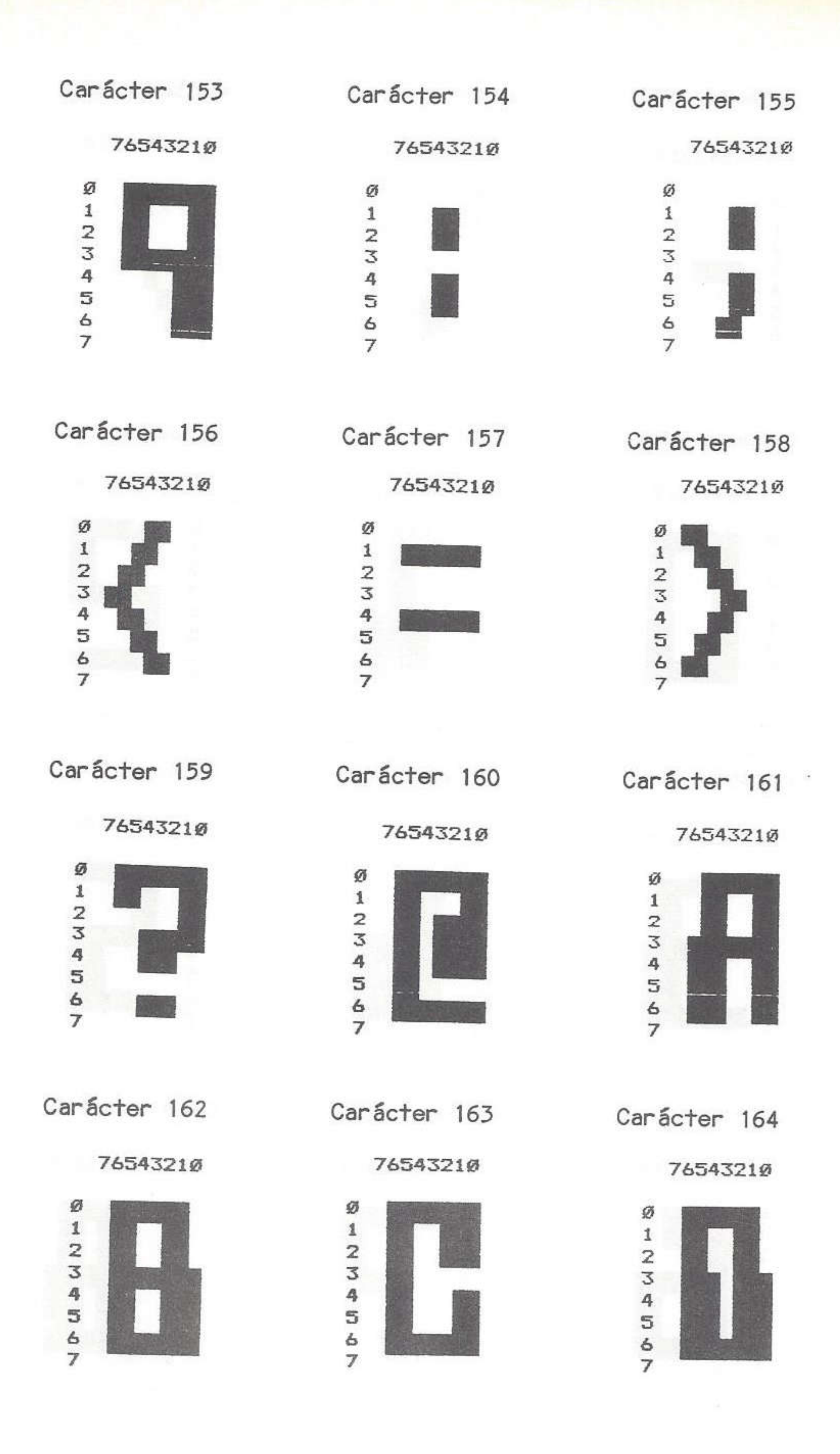
76543210

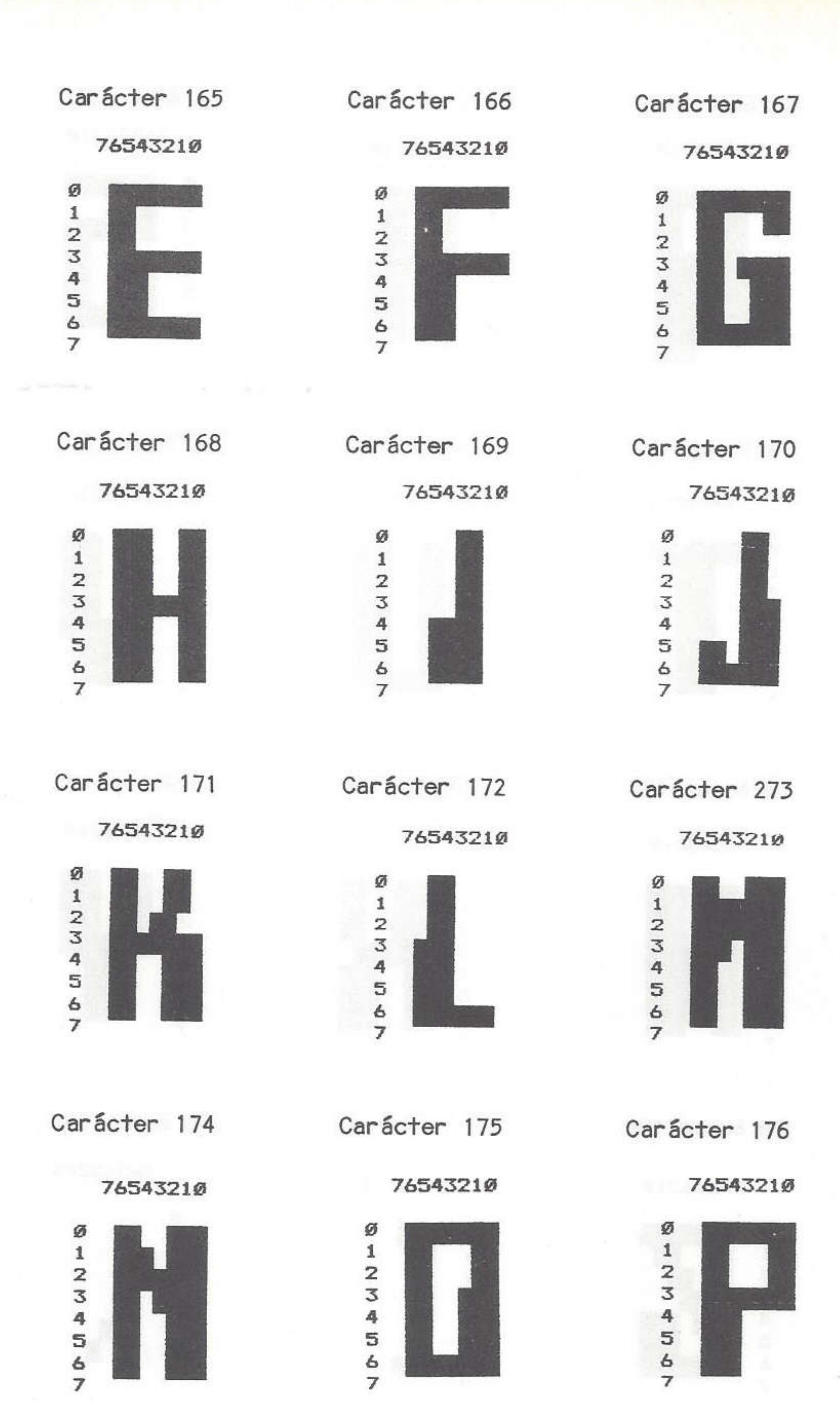




Carácter 129
76543210
Ø 1 2 3 4 5 6 7
Carácter 130
76543210
Ø 1 2 3 4 5 6 7
Carácter 131
76543210
Ø 1 2 3 4 5 6 7
Carácter 132
76543210
Ø 1 2 3 4 5 6 7
Carácter 133
76543210
Ø 1 2 3 4 5 6 7
Carácter 134
76543210
Ø 1 2 3 4 5 6 7
Carácter 135
76543210
Ø 1 2 3 4 5 6 7
Carácter 136
76543210
Ø 1 2 3 4 5 6 7
Carácter 137
76543210
Ø 1 2 3 4 5 6 7
Carácter 138
76543210
Ø 1 2 3 4 5 6 7
Carácter 139
76543210
Ø 1 2 3 4 5 6 7
Carácter 140
76543210
Ø 1 2 3 4 5 6 7


Carácter 141
76543210
Ø 1 2 3 4 5 6 7
Carácter 142
76543210
Ø 1 2 3 4 5 6 7
Carácter 143
76543210
Ø 1 2 3 4 5 6 7
Carácter 144
76543210
Ø 1 2 3 4 5 6 7
Carácter 145
76543210
Ø 1 2 3 4 5 6 7
Carácter 146
76543210
Ø 1 2 3 4 5 6 7
Carácter 147
76543210
Ø 1 2 3 4 5 6 7
Carácter 148
76543210
Ø 1 2 3 4 5 6 7
Carácter 149
76543210
Ø 1 2 3 4 5 6 7
Carácter 150
76543210
Ø 1 2 3 4 5 6 7
Carácter 151
76543210
Ø 1 2 3 4 5 6 7
Carácter 152
76543210
Ø 1 2 3 4 5 6 7


Carácter 153
76543210
Ø
1
2
3
4
5
6
7
Carácter 154
76543210
Ø
1
2
3
4
5
6
7
Carácter 155
76543210
Ø
1
2
3
4
5
6
7
Carácter 156
76543210
Ø
1
2
3
4
5
6
7
Carácter 157
76543210
Ø
1
2
3
4
5
6
7
Carácter 158
76543210
Ø
1
2
3
4
5
6
7
Carácter 159
76543210
Ø
1
2
3
4
5
6
7
Carácter 160
76543210
Ø
1
2
3
4
5
6
7
Carácter 161
76543210
Ø
1
2
3
4
5
6
7
Carácter 162
76543210
Ø
1
2
3
4
5
6
7
Carácter 163
76543210
Ø
1
2
3
4
5
6
7
Carácter 164
76543210
Ø
1
2
3
4
5
6
7


Carácter 165
76543210
Ø
1
2
3
4
5
6
7
Carácter 166
76543210
Ø
1
2
3
4
5
6
7
Carácter 167
76543210
Ø
1
2
3
4
5
6
7
Carácter 168
76543210
Ø
1
2
3
4
5
6
7
Carácter 169
76543210
Ø
1
2
3
4
5
6
7
Carácter 170
76543210
Ø
1
2
3
4
5
6
7
Carácter 171
76543210
Ø
1
2
3
4
5
6
7
Carácter 172
76543210
Ø
1
2
3
4
5
6
7
Carácter 273
76543210
Ø
1
2
3
4
5
6
7
Carácter 174
76543210
Ø
1
2
3
4
5
6
7
Carácter 175
76543210
Ø
1
2
3
4
5
6
7
Carácter 176
76543210
Ø
1
2
3
4
5
6
7

Carácter 177

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 178

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 179

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 180

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 181

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 182

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 183

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 184

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 185

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 186

7654321Ø

Ø 1 2 3 4 5 6 7

Carácter 187

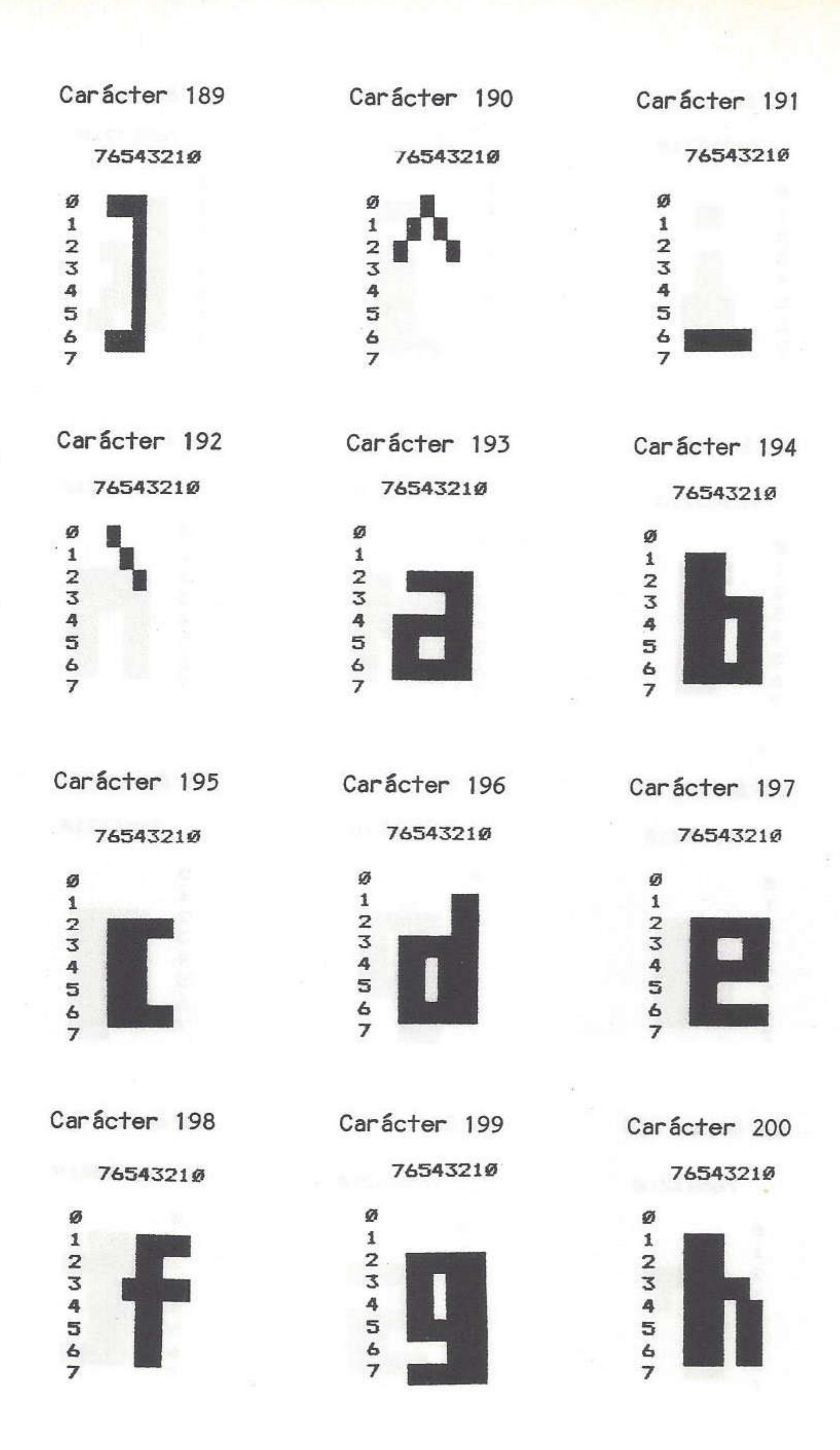
7654321Ø

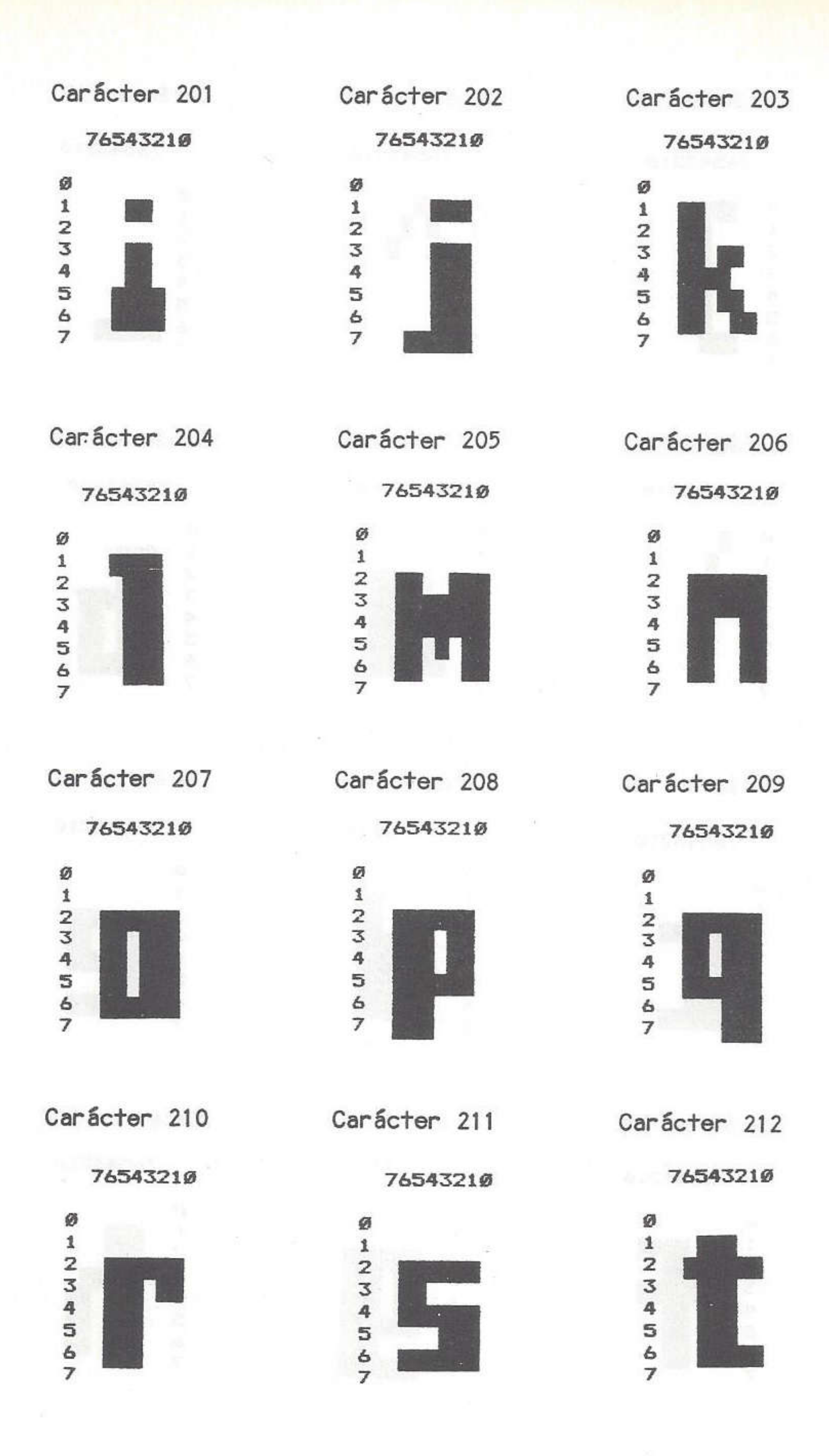
Ø 1 2 3 4 5 6 7

Carácter 188

7654321Ø

Ø 1 2 3 4 5 6 7


Carácter 189
76543210
0 1 2 3 4 5 6 7
Carácter 190
76543210
0 1 2 3 4 5 6 7
Carácter 191
76543210
0 1 2 3 4 5 6 7
Carácter 192
76543210
0 1 2 3 4 5 6 7
Carácter 193
76543210
0 1 2 3 4 5 6 7
Carácter 194
76543210
0 1 2 3 4 5 6 7
Carácter 195
76543210
0 1 2 3 4 5 6 7
Carácter 196
76543210
0 1 2 3 4 5 6 7
Carácter 197
76543210
0 1 2 3 4 5 6 7
Carácter 198
76543210
0 1 2 3 4 5 6 7
Carácter 199
76543210
0 1 2 3 4 5 6 7
Carácter 200
76543210
0 1 2 3 4 5 6 7


Carácter 201
76543210
0 1 2 3 4 5 6 7
Carácter 202
76543210
0 1 2 3 4 5 6 7
Carácter 203
76543210
0 1 2 3 4 5 6 7
Carácter 204
76543210
0 1 2 3 4 5 6 7
Carácter 205
76543210
0 1 2 3 4 5 6 7
Carácter 206
76543210
0 1 2 3 4 5 6 7
Carácter 207
76543210
0 1 2 3 4 5 6 7
Carácter 208
76543210
0 1 2 3 4 5 6 7
Carácter 209
76543210
0 1 2 3 4 5 6 7
Carácter 210
76543210
0 1 2 3 4 5 6 7
Carácter 211
76543210
0 1 2 3 4 5 6 7
Carácter 212
76543210
0 1 2 3 4 5 6 7

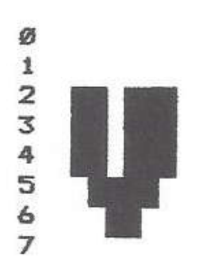
Carácter 213

76543210



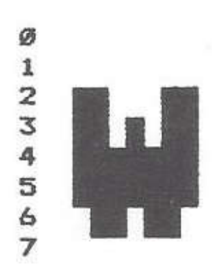
Carácter 214

76543210



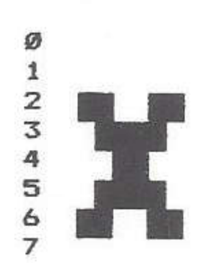
Carácter 215

76543210



Carácter 216

76543210



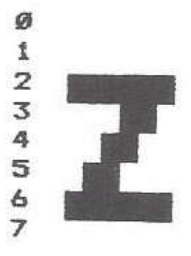
Carácter 217

76543210



Carácter 218

76543210



RESPUESTA
COMERCIAL

F.D. Autorización 6975
(B.O. de Correos N.º 80 de 26-7-85)

HOJA PEDIDO
DE LIBRERIA

NO NECESITA
SELLOS

A franquear
en destino

FERRE MORET, S.A.

Apartado N.º 551. F.D.
08080 BARCELONA

RESPUESTA
COMERCIAL

F.D. Autorización 6975
(B.O. de Correos N.º 80 de 26-7-85)

HOJA PEDIDO
DE LIBRERIA

NO NECESITA
SELLOS

A franquear
en destino

FERRE MORET, S.A.

Apartado N.º 551. F.D.
08080 BARCELONA

Puesta al día de datos

EDITORIAL FERRER MORET, S.A. mantiene vivo y amplía el contenido informativo de sus libros y programas, mediante el envío de un servicio de puesta al día, junto con una síntesis noticiosa de la actualidad y perspectivas de la realidad informática española.
Agradecemos cualquier sugerencia o crítica que desee formular y que nos ayude a mejorar las ediciones. Muchas gracias.

¿Qué añadiría?
....

....

¿Qué suprimiría?
....

....

Observaciones
....

....

Título del libro
....

Nombre
....

Dirección .... Tfno.
....

Código Postal y Población .... Provincia
....

UN SERVICIO GRATUITO

Información

FERRE MORET, S.A. cuenta con un amplio fondo de libros y Software y mantiene un servicio de información por correo sobre las novedades que edita.
Agradecemos nos indique los temas que representan para Vd. mayor interes.

Libros	☐ ATARI	☐ MSX	☐ AMSTRAD
	☐ COMMODORE	☐ LENGUAJES	☐ APPLE
	☐ SINCLAIR	☐ IBM	☐ SOFTWARE

Si está interesado en recibir alguno de estos servicios, rellene y envíe la tarjeta correspondiente; no necesita franqueo. Muchas gracias.

DESEO RECIBIR EL LIBRO ....

EL PROGRAMA ....

☐ Adjunto cheque ☐ Contra reembolso
....

Nombre
....

Dirección .... Tfno.
....

Código Postal y Población .... Provincia
....

UN SERVICIO GRATUITO

EL CONTENIDO

Este libro empieza allí donde termina el manual de instrucciones de su ordenador MSX:

En primer lugar se analizan detenidamente todos los comandos de gráficos y sonido del Basic MSX (particularmente los más complejos, como el VDT, SOUND, BASE). Cada comando va acompañado de eficaces programas de ejemplo (más de 100), algunos de los cuales puede incluir en sus propios programas. A continuación se dan consejos y trucos sobre la forma de sacar el máximo provecho del sonido y del gráfico de su ordenador MSX.

Además, en este libro también encontrará un gran número de programas de primera fila. (Continuación del libro Programas y Utilidades de DATA BECKER), como por ejemplo:

- Generador de caracteres
- Gráficos de pantallas de texto
- Editor de Play y SOUND con confección automática de subprogramas
- MISTER MINER y FRETTY, dos juegos de primera línea con varias pantallas SPRITES de varios colores

ESTE LIBRO LO HA ESCRITO:

Rainer Luers, maestro de profesión, técnico de ordenadores de una de las más importantes empresas de almacenes y autor experimentado de varios libros de DATA BECKER (CPC-464 Programas Basic, MSX Programas y Utilidades y MSX para principiantes).

ISBN 84-86437-25-3